SWARMING RECONNAISSANCE USING UNMANNED AERIAL VEHICLES IN A PARALLEL DISCRETE EVENT SIMULATION

THESIS

Joshua J. Corner, Captain, USAF

AFIT/GCE/ENG/04-01

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCE/ENG/04-01

Swarming Reconnaissance Using Unmanned Aerial Vehicles in a Parallel Discrete Event Simulation

THESIS

Presented to the Faculty of the
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Engineering

Joshua J. Corner, B.S.E.E
Captain, USAF

March, 2004

AFIT/GCE/ENG/04-01

Swarming Reconnaissance Using Unmanned Aerial Vehicles in a Parallel Discrete Event Simulation

Joshua J. Corner, B.S.E.E

Captain, USAF

Approved:

/signed/	5 Mar 04
Dr. Gary B. Lamont (Chairman)	Date
/signed/	11 Mar 04
Dr. Meir Pachter (Member)	Date
/signed/	12 Mar 04
Dr. Gilbert L. Peterson (Member)	Date
/signed/	5 Mar 04
Major Brian A. Kadrovach, PhD (Member)	Date

Acknowledgements

Praise the Lord!!! He has brought me through this challenging season. Hallelujah Thine the glory! Great things He hath done. Listen to what the Scripture saith:

Psalms 107:8 (KJV) Oh that men would praise the LORD for his goodness, and for his wonderful works to the children of men!

Psalms 107:15 (KJV) Oh that men would praise the LORD for his goodness, and for his wonderful works to the children of men!

Psalms 107:21 (KJV) Oh that men would praise the LORD for his goodness, and for his wonderful works to the children of men!

Psalms 107:31 (KJV) Oh that men would praise the LORD for his goodness, and for his wonderful works to the children of men!

And so I praise my Creator, my Redeemer, my Savior, my Tower of Refuge, my Comforter, my Lord, my God, my Strength, my All in all. For He alone is worthy. Without Him I can do nothing. Glory to God in the highest. He gave me the desire of my heart and then provided me with the ability to see it to completion.

The Lord saw fit to use several people along the way to encourage, to guide, to direct, to comfort, to instruct, and to challenge me. The first among those who have helped includes my beloved wife and our two beautiful boys. What a joy to have family share this experience. Next, my advisor has been an excellent person to study under. Next, my friends in the GCE-04 class– all five of them–have made the long journey interesting and full of laughter. Next, the folks in AFRL/IFTC and AFRL/SNZW have provided expertise and tools that added significantly to this work. Next, that lone PhD student under the same advisor has helped in numerous ways. Next to last, the 2Lt Matlab expert who worked down the hill from the main campus contributed hours of his time. And last there are many more at my church who have given support along the way. All of you are appreciated greatly.

Praise Jesus Christ my Lord!

Joshua J. Corner

Contents

Page

List of Figures

List of Tables

List of Abbreviations

Abbreviation		Page
UAV	Unmanned Aerial Vehicle	1
DOD	Department of Defense	2
IDAL	Integrated Demonstrations and Applications Laboratory	4
DOD	Department of Defense	8
UCAV	Unmanned Combat Aerial Vehicle	12
MAV	Micro Air Vehicles	12
DARPA	Defense Advanced Research Projects Agency	13
DSNs	Distributed Sensor Networks	15
OSI	Open Systems Interconnect	17
PDES	Parallel Discrete Event Simulation	27
HPC	High Performance Computing	30
METT-T	mission, enemy, terrain, troops, time	32
BDA	Battle Damage Assessment	34
MOE	Measures of Effectiveness	34
MOP	Measures of Performance	35
LVT	local virtual time	36
LP	logical process	36
lcc	local causality constraint	36
SPEEDES	Synchronous Parallel Environment for Emulation and Discrete-Event Simulation	58
BTW	Breathing Time Warp	59
DDM	Data Distribution Management	60
MFC	Microsoft Foundation Class	62
CPUs	Central Processing Units	75
AFIT	Air Force Institute of Technology	110
MIMD	Multiple Instruction Multiple Data	110

Abbreviation		Page
UMA	Uniform Memory Access	110
NUMA	Non-Uniform Memory Access	110

Abstract

Current military affairs indicate that future military warfare requires safer, more accurate, and more fault-tolerant weapons systems. Unmanned Aerial Vehicles (UAV) are one answer to this military requirement. Technology in the UAV arena is moving toward smaller and more capable systems and is becoming available at a fraction of the cost. Exploiting the advances in these miniaturized flying vehicles is the aim of this research.

How are the UAVs employed for the future military? The concept of operations for a micro-UAV system is adopted from nature from the appearance of flocking birds, movement of a school of fish, and swarming bees among others. All of these natural phenomena have a common thread: a global action resulting from many small individual actions. This "emergent behavior" is the aggregate result of many simple interactions occurring within the flock, school, or swarm. In a similar manner, a more robust weapon system uses emergent behavior resulting in no "weakest link" because the system itself is made up of simple interactions by hundreds or thousands of homogeneous UAVs. The global system in this research is referred to as a swarm. Losing one or a few individual unmanned vehicles would not dramatically impact the "swarms" ability to complete the mission or cause harm to any human operator. Swarming reconnaissance is the emergent behavior of swarms to perform a reconnaissance operation.

An in-depth look at the design of a reconnaissance swarming mission is studied. A taxonomy of passive reconnaissance applications is developed to address feasibility. Evaluation of algorithms for swarm movement, communication, sensor input/analysis, targeting, and network topology result in priorities of each model's desired features. After a thorough selection process of available implementations, a subset of those models are integrated and built upon resulting in a simulation that explores the innovations of swarming UAVs. Visualization of the swarm is accomplished through a post-processing visual system as well as a near real-time system.

Exploration of these concepts is accomplished through a high performance computing parallel discrete event simulation. That platform is used as the test bed for swarming reconnaissance. After development of the system, several experiments are designed, tested, and analyzed for ef-

ficiency and effectiveness. Results indicate that swarming reconnaissance is a feasible option for our future military.

Swarming Reconnaissance Using Unmanned Aerial Vehicles

in a

Parallel Discrete Event Simulation

1. Introduction and Overview

This chapter provides a high-level picture of the research conducted. Descriptions of key concepts, goals, and associated sponsoring organizations are presented. The research approach is outlined including assumptions and risks. The chapter ends with an overall layout of the thesis.

1.1 Problem Statement

Current military affairs indicate that future military warfare requires safer, more accurate, and more fault-intolerant weapons systems [35][20]. With the war in IRAQ claiming the attention of the television's viewing audience of the United States, there is an overwhelming desire in the eyes of both the citizens and military leaders for a safer, more accurate long range strike ability. Unmanned Aerial Vehicles (UAV) are one answer to this military requirement [14][30][67][37]. At least 11 types of UAVs were committed to Operation Iraqi Freedom [95]. Technology in the Unmanned Aerial Vehicles arena is moving toward smaller and more capable systems and is becoming available at a fraction of the cost. How does one exploit this innovative technology in attempting to satisfy these future requirements?

1.2 Key Concepts

1.2.1 Unmanned Aerial Vehicles. The ability to use cooperative autonomous vehicles to perform a wartime mission is an important application of the future requirements. Those autonomous vehicles are Unmanned Aerial Vehicles (UAV) which are mobile airborne machines that do not require an on-board human operator. Typically they are controlled by a remote operator or autonomous control logic. They have a history stretching back to the Civil War era. A patent for an unmanned aerial bomber balloon was issued to Charles Perley of New York City in February

of 1863 [78]. While Perley's UAV was never used in combat operations it demonstrated the first application of unmanned aerial vehicles to military operations in the United States.

The Department of Defense (DOD) defines UAVs as "powered, aerial vehicles that do not carry a human operator, use aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload" [13]. All four major United States (US) military branches have UAVs in their inventory. The military effectiveness of UAVs in recent conflicts such as Iraq (2003), Afghanistan (2001), and Kosovo (1999) has opened the eyes of many to the advantages provided by unmanned aircraft. They are used across a range of sensor-focused operations from high-fidelity real-time reconnaissance missions to battle damage assessment to armed attack missions. This research work focuses on the use of UAVs to perform a reconnaissance mission.

The autonomous UAVs considered in this research are comparable in size to a bumblebee. Great strides are being made in miniaturization of electronic and electro-mechanical systems. The Office of Naval Research originated the idea of a robofly in 1998. Current expectations indicate that it will be airborne by 2004. Supporting this research is a hefty bankroll of $2.5 million. Investigators at the University of California-Berkeley have the challenge of exploiting the sophisticated flight control system of flies for two complementary purposes: i) to identify simple yet robust flight control algorithms for use in autonomous flying devices, and ii) to identify means of externally controlling the flight trajectory of real flies [70]. The robofly's projected weight is about 43 milligrams-roughly the weight of a fat housefly. It will zip along at 3 meters (about 10 feet) per second and have a range of about 2 kilometers (about 1 1/4 miles). A significant achievement in the course of this project occurred in 1999 when biologist Michael Dickinson discovered that insects use a complex choreography of three different wing motions to generate lift and thrust. Ron Fearing, a UC Berkeley electrical engineer, is currently designing wings capable of mimicking those movements while Dickinson is studying how flies navigate with reaction speeds that allow them to change course in just 30-thousandths of a second [93]. Figure 1 shows an artist's conceptual drawing of the completed micro-mechanical flying insect being developed at Berkeley. Outside the United States is the Seiko Epson Corporation's "Micro Flying Robot" (uFR) [77]. This uFR demonstrated its micromechatronics technology in November 2003 at the International Robot Exhibition and is known as the world's smallest flying prototype micro robot [77]. These

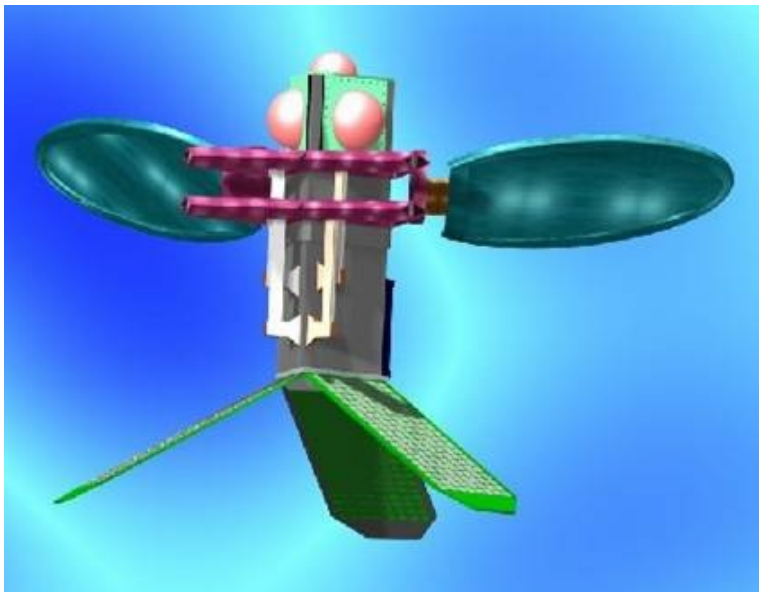


Photo courtesy R.Fearing/UC-Berkely

Figure 1 Conceptual Drawing of Completed Robofly

future miniaturized UAVs equipped with wireless communications elements and multiple types of sensors provide sensor data in numerous environments including those unsuitable for traditional sensor systems.

1.2.2 Swarming. A large number of these devices can work together like a swarm of insects or a flock of birds to provide high fidelity information on a near or real time basis. Natural swarming is an "emergent behavior" which is the aggregate of many simple interactions occurring within the flock, school, or swarm. The global system in this research is referred to as a swarm. Losing one or a few individual unmanned vehicles would not dramatically impact the "swarms" ability to complete the mission or cause harm to any human operator. It has no "weakest link" because the system itself is made up of simple interactions by hundreds or thousands of homogeneous UAVs. Swarming or emergent behavior systems present a unique implementation concept for a sensor system with a large number of individual sensors. Swarm behavior, like that seen in bee swarms or flocks of birds provides a stable organization of sensor platforms that is flexible, able to adjust rapidly to changing environmental conditions. For example, should an item of high interest show up in the sensor field, a smaller sub-swarm could break off and perform a

higher-resolution recon operation and then rejoin the main swarm to relay and process the data. The result is a more robust, flexible, and efficient weapon system that uses emergent behavior.

1.3 Sponsors

The research is sponsored by the Information Directorate, Air Force Research Laboratory (AFRL), Wright Patterson Air Force Base, Ohio and Rome Labs in Rome, New York. The mission of the Information Directorate is "the advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to air and space systems to meet war fighter needs." This mission is accomplished through research and development of embedded information systems capable of delivering timely information about the battle space to the war fighter while surviving threats. The research discussed in this thesis supports this mission by developing an evaluation model for a swarm of mobile sensor platforms, moving through the battle space autonomously, and transmitting high-resolution fused data to the war fighter. Specific points of contact are Dr. Bob Ewing (AFRL/IFTA), Bob Smith (AFRL/VACC), and Dr. Douglas Holzhauer (AFRL/IFTC).

In addition, the AFRL sensors applications and demonstrations division (AFRL/SNZ) support this research. The specific branch in that division that has an interest is the electronic warfare branch (AFRL/SNZW.) Their mission is "Conducts advanced development field and flight test demonstrations, and supporting risk reduction simulation/evaluations of radio frequency (RF); electro-optical (EO)/infrared (IR); and offensive command and control warfare (C^2W) electronic warfare (EW) sensor systems and subsystems for all AF air and space vehicles. Demonstrations encompass one or both functions of electronic support (ES) inclusive of surveillance/ reconnaissance, threat warning, and identification, and electronic attack (EA) of RF/EO/IR threat systems and associated C^2/reference systems." The specific area connected to this research is their Integrated Demonstrations and Applications Laboratory (IDAL.) It is a man / hardware- in- the- loop simulation facility for maturing advanced sensor technologies by subjecting these technologies to multiple realistic combat situations. The specific point of contact is Mike Foster (AFRL/SNZW).

1.4 Goal, Objectives, and Approach

The research goal is to "provide an understanding of how UAV technology can help develop an accurate and fault tolerant distributed sensor swarm satisfying future military system requirements." In this effort, there are two main objectives:

1. Develop, parallelize, and evaluate a swarm model simulation system
2. Evaluate performance (effectiveness and efficiency) of a swarm reconnaissance mission using this simulation

1.4.1 Objective 1: Parallel Swarm Simulation. The first objective is developing and parallelizing a swarm simulation system. A parallel simulation enables inclusion of higher fidelity simulation models without increased resource requirements resulting in a more accurate representation of the future weapon system. The "system" includes supporting infrastructure to include all pertaining models that are necessary to simulate a sensor swarm reconnaissance mission. The simulation benefits from advantages of distributed processing such as task decomposition, decreased execution time, concurrent processing, and problem simplification. Simulating a swarming system involves many highly detailed models that include sensor node capability, node movement, node communications, system environment, targets, terrain, search strategy and emergent swarm behavior. Specific steps that mark progress toward completing this objective include the following sub-objectives:

1.1 Development of a swarm simulation model

1.2 Accurate parallelization of the simulation model

1.3 Optimization of the parallel simulation implementation (efficiency)

1.4 Evaluation of the efficiency of the parallelized simulation system

1.4.2 Objective 2: Evaluate Swarming Reconnaissance Mission Effectiveness. The second objective, mission effectiveness, is to present an end to end analysis of the efficacy of using swarming to produce a desired military outcome. One war-time mission is considered– reconnaissance. According to military doctrine, reconnaissance information is used to "provide

critical intelligence that is vital to the shaping of the battle space [7]"; therefore, it is an appropriate selection for impacting current military strategy. A ideal reconnaissance scenario is made of a micro-UAV model, swarm movement algorithms, on-board sensor emulation, autonomous data acquisition , sensor fusion, and a network communication model. All these characteristics are considered when evaluating the effectiveness of a swarm-based reconnaissance mission. Specific steps that mark progress toward completing this objective include the following sub-objectives:

2.1 Development of supporting models

2.2 Define specific reconnaissance scenarios

2.3 Define effectiveness criteria for each scenario (metrics)

2.4 Simulate each reconnaissance scenario for mission effectiveness

In order to produce a qualitative solution the models representing swarm movement, communication, data fusion, and target environment must have an acceptable level of fidelity. This is accomplished through defining an acceptable fidelity level for each model.

1.4.3 Assumptions. The parallel simulation development is an iterative process with small increments through which all the models can be integrated. It is assumed that the UAVs are modeled in two dimensions for simplicity. This application can be extended to three dimensions where the UAVs are flying at a fixed altitude. Ground targets are the interest items to be discovered by the reconnaissance mission.

1.5 Thesis Overview

Researching a subject is not chronological nor sequential; however, for structure this document maintains sequential format. Chapter 2 defines and develops concepts that are critical to making effective design decisions about UAVs, distributed sensor processing, ad-hoc networks, swarming, reconnaissance, and parallel discrete event simulation. Chapter 3 takes that context and applies high level design strategies toward the development of a swarming reconnaissance model based in a parallel discrete event simulation running in a high performance computing environment. Chapter 4 provides the transition from high level concepts to reality through selecting the simulator and then implementing the model. Chapter 5 uses that implementation as a tool set for

validating the objectives of this research by designing related experiments. Chapter 6 discusses the execution of those experiments and shows how the results are analyzed. The final chapter makes conclusions according to analyzed results and relates them back to the objectives that are presented this chapter.

2. Research Context

A parallel swarm simulation and reconnaissance application touch many subject areas. The details presented in this chapter establish the context in which this research is conducted. It presents relevant research developments in unmanned aerial vehicles, distributed sensor networks/processing, swarming, parallel discrete event simulation, and the subject of reconnaissance, along with a brief lower level review of the pertinent subjects contained within each major division.

2.1 Unmanned Aerial Vehicles

Unmanned Aerial Vehicles have been referred to in many ways: remotely piloted vehicle, drone, robot plane, aerial target and pilot-less aircraft are some examples. Most often called UAVs, they are defined by the Department of Defense (DOD) as "powered, aerial vehicles that do not carry a human operator, use aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload" [13]. The military effectiveness of UAVs in recent conflicts such as Iraq (2003), Afghanistan (2001), and Kosovo (1999) has opened the eyes of many to the advantages provided by unmanned aircraft.

2.1.1 Historical Development.

Unmanned aerial vehicles have a history stretching back to the Civil War era. Although none of the balloon bombs (a.k.a UAVs) were used during the Civil War, a patent for an unmanned aerial bomber balloon was issued to Charles Perley of New York City in February 1863. This device consisted of a hot-air balloon, explosives, and a timer as shown in Figure 2. The closed view exhibits the bomb in the basket. The open view shows the bomb falling out of the bottom. When the timer expired it would trip a hammer on a cylinder which would eject a hinge pin. As the pin ejected, it also ignited the bombs fuse. At this point, the hinged bottom of the basket would open and release the bomb. The most important problem with his invention was that the weapon could only be used when the wind was blowing in the direction of the enemy. Perley's idea was never taken seriously and therefore was not implemented [78].

Although limited progress in unmanned technology continued for the next 100 years, serious interest in UAVs as operational force multipliers has only awakened in the last three decades. The editor of Jane's Unmanned Aerial Vehicles writes:

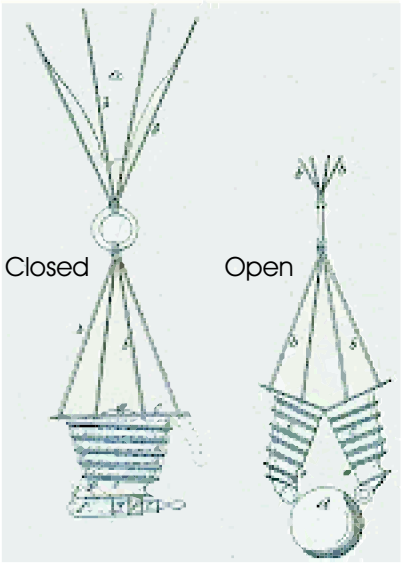


Figure 2 Perley's Drawings of Unmanned Bomber

> The catalyst was the Vietnam War, in which the outstanding record of Teledyne Ryan's AQM-34 'Lightning Bugs' proved once again how a hasty improvisation in the heat of battle can often turn out to be a match winning combination. With the ending of that unfortunate conflict, the 1970s seemed to promise a new era in which unmanned aircraft would quickly evolve into an accepted form of modern military equipment. The aerospace industry, especially in the USA, had a field day. Many bizarre designs were flight-trialled. Also, it must be said, many highly promising ones went to the wall for lack of political or military support, but though the technological interest remained high, the expected orders never came. Instead, the focus of activity shifted to the Middle East, where the sheer survival of national sovereignty motivated Israel to develop and use its own first generation of unmanned decoys and surveillance UAVs. [75]

The first nation acknowledged to have made UAVs a standard was Israel. They reasoned that it was comparatively less costly to risk losing a UAV rather a pilot and a multi-million-dollar plane. What the Israeli Air Force did was use remotely piloted Scout vehicles to fool Syrian radar sites into activating their radars. During the Bekaa Valley War, the Israeli bombers used this technique to locate and destroy 19 missile sites achieving air superiority over Syria [45].

The success of Israel's tactical use of UAVs during operations in Lebanon in 1982 motivated then-Navy Secretary John Lehman to acquire a UAV capability for the Navy. Interest also grew in other parts of the Pentagon, and the Reagan Administration's FY1987 budget submission included

increased UAV procurement [13]. The acquired Pioneer was procured initially by the Navy in 1985 and subsequently used in over 300 missions in the Persian Gulf during 1990 and 1991. Israel and the United States are not the only nations interested in UAVs. Europe, Germany, France, Russia, Norway, Iraq, Turkey, Asia, Pakistan, Japan, North & South Korea, and China all have active UAV programs [109].

The role of the unmanned vehicle has expanded beyond what Perley envisioned with his balloon bomber to include use as decoys for the Israeli Air Force. Thus as technology continually expanding so does the feasibility of new roles for UAVs. This trend was recognized by the DOD in 1995 when it started the requirement for annual reporting on UAV progress in technology.

2.1.2 Current Development of UAVs in United States Military . There are currently five major UAVs in the U.S. inventory [13]: the Navy and Marine Corps's Pioneer, the Air Force's Global Hawk and Predator, and the Army's Hunter and Shadow UAVs. Their functions range from high-fidelity real-time reconnaissance missions to battle damage assessment to armed attack missions.

UAVs have been labeled as transformational technologies that can change how wars are fought and won. President Bush used the UAV as an example of a technology that is changing the face of the battlefield during his speech to the Citadel in December 2001. Speaking of the conflict in Afghanistan, Bush stated:

> The Predator is a good example. This unmanned aerial vehicle is able to circle over enemy forces, gather intelligence, transmit information instantly back to commanders, then fire on targets with extreme accuracy. Before the war, Predator had skeptics, because it did not fit the old ways. Now it is clear the military does not have enough unmanned vehicles. We're entering an era in which unmanned vehicles of all kinds will take on greater importance.[18]

The Navy and Marine Corp's Pioneer is a short range, tactical UAV. This imagery intelligence platform was acquired in 1985 and is made by Pioneer UAV, Incorporated. It can fly as high as 12,000 feet with a range around 185 kilometers. While the Pioneer received most of its acclaim during the 300+ combat reconnaissance missions during Operation Desert Shield and Desert Storm in 1990-91, it has supported every major U.S. contingency operation during the first 10 years of its service [82].

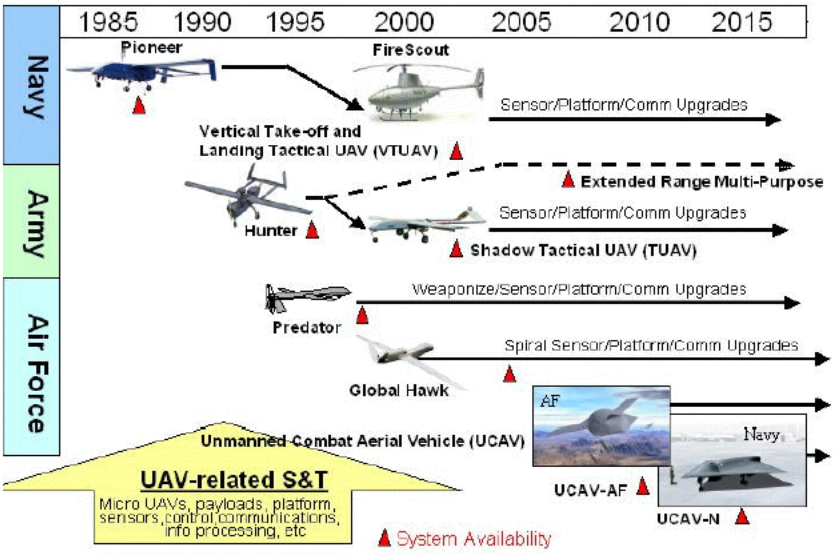


Figure 3 UAV Acquisition Status

Source: DODBackground Briefing on UAVs, [http://www.defenselink.mil], October 31, 2001. *Note: Navy includes Navy and Marine Corps; Navy VTUAV Firescout program canceled in 2001, causing the Navy Pioneer program to be extended through 2010.

Both the Air Force's Global Hawk and Predator are from the endurance class of UAVs. The Hawk notably was called into service earlier than planned to aid in the war on terrorism [82]. It can fly up to an altitude of 65,000 feet with a range of 3000 nautical miles. A possible day for the Hawk might involve covering 40,000 square miles providing imagery with resolutions down to 3 feet. Like the Hawk, the Predator also is an intelligence gathering platform but at lower altitude (26,000 feet) and at a shorter range of operations (400 nm) [106]. Though originally designed for reconnaissance activities, several Predators have been armed with hellfire missiles and were successfully launched against SUV convoys, buildings, and even human targets [89].

The Army's Hunter UAV is an Israeli multi-role short range tactical UAV. It flies up to an altitude of 16,000 feet and has a range up to 125 kilometers. The mission of the Hunter is day and night reconnaissance, intelligence, surveillance, and target acquisition for Corps Commanders [82]. Although development of this UAV stopped abruptly after 20 vehicle crashes during initial testing, there still remains 43 units in DOD inventory [13].

2.1.2.1 Combat UAV Development. While the common military mission for Unmanned Aerial Vehicles is passive intelligence gathering, aggressive UAV variants are also being

developed. The Unmanned Combat Aerial Vehicle (UCAV) is the title given to this variant. It has an implied purpose of aggressively engaging a target in order to disable or destroy it. With congressional budget support this combat UAV looks promising [39].

In [67], the current status of two UCAVs is presented: the X-45A and the X-47B. Boeing has developed the X-45A air vehicle with one weapons bay and an avionics suite. This UCAV has four levels of autonomy all with varied levels of participation from a ground controller with the fourth level being fully autonomous. An attractive feature of these unmanned combat vehicles is the low purchase and maintenance costs when compared to current systems. One unique feature about the X-47B that is built by Northop Grumman includes self-aware software. This software has the ability to monitor itself and react when a part malfunctions by switching to a backup. Both highlighted features of these UCAVs are hinging arguments not only for UCAVs but also a swarm system of UAVs. The lower overall cost allows for redundancy, survivability, increased mission effectiveness by not requiring pilot interfaces or training. These low-risk vehicles also have benefits of long-term storage capability and reuse. The self-adapting capability is not unique to the X-47B, but a swarm also exhibits similar behavior because there it has no single point of failure.

2.1.3 UAV Development: State of the Art. Current UAV technology trends are toward micro-UAVs, commonly called Micro Air Vehicles (MAV), perhaps the 21st Century war-fighters. Their dimensions are measured in centimeters and inches rather than meters or feet and their weight in grams rather than pounds. To appreciate the scale implications, the Navy's Pioneer UAV has a Reynolds number (a measure of size multiplied by speed) on the order of $400,000$ while an average Micro Air Vehicle only $6,000$. This low number is significant because of the fundamental shift in physical behavior at MAV scales and speeds–an environment more common to the smallest birds and insects. This means that the slightest turbulence can have a serious impact on the flight stability for these Micro Air Vehicles. But their size is hardly a disadvantage.

Up-to-date intelligence on the battlefield for a front-line soldier is scarce. Micro Air Vehicles can enable combat troops to see over the next hill or behind a nearby building in an urban setting. The Department of Defense is trying to help ground troops through the use of a MAV called the Black Widow. Technical evaluations conducted at the Massachusetts Institute of Technology's

Lincoln Laboratories in Lexington and the Naval Research Laboratory in Washington, D.C., have concluded that the concept is workable.

In 1996, the DOD's Defense Advanced Research Projects Agency (DARPA) invested $35 million over a period of 4 years for nine innovative research contracts for the development of operating concepts and to demonstrate flight-enabling technologies [105]. Two years later 4 of the 9 progressed into Phase 2 contracts. DARPA continues funding MAV projects with a specified package size of six inches. The designs for MAVs range from mimicking biological flapping wings to micro version helicopters to a scientific robofly. To highlight the latest research in these areas two projects are reviewed: Entomopter and robofly. For a more complete list of current research projects in Micro Air Vehicles see [40].

The Entomopter Project is a multi-mode (flying/crawling) mechanical insect based on a reciprocating chemical muscle which is capable of generating autonomic wing beating from a chemical energy source. Therefore it achieves abnormally high lift by its rapidly flapping wings. This DARPA-funded man-made insect is also capable of steered flight through differential lift enhancement on the wings that achieve roll. NASA has noted this MAV has the unique ability to fly on the planet Mars where a fixed wing aircraft would have to fly over 250 mph just to stay aloft in the rarefied atmosphere. The U.S. Patent office granted patents for the overall system concept in 2000 as well as the propulsion system in 2002. Because this technology completed its concept demonstration phase, current work is progressing at the Georgia Institute of Technology to develop the wings for the Mars Entomopter [71].

The Office of Naval Research originated the idea of a robofly in 1998. Current expectations indicate that it will be airborne by 2004. Supporting this research is a hefty bankroll of $2.5 million. Investigators at the University of California-Berkeley have the challenge of exploiting the sophisticated flight control system of flies for two complementary purposes: i) to identify simple yet robust flight control algorithms for use in autonomous flying devices, and ii) to identify means of externally controlling the flight trajectory of real flies [70]. The robofly will weigh about 43 milligrams-roughly the weight of a fat housefly. It will zip along at 3 meters (about 10 feet) per second and have a range of about 2 kilometers (about 1 1/4 miles). A significant achievement in the course of this project occurred in 1999 when biologist Michael Dickinson discovered that insects use a complex choreography of three different wing motions to generate lift and thrust. Ron

Fearing, a UC Berkeley electrical engineer, is currently designing wings capable of mimicking those movements while Dickinson is studying how flies navigate with reaction speeds that allow them to change course in just 30-thousandths of a second [93].

Delivery of MAVs has also been the subject of current research efforts. While it is not feasible to expect MAVs to travel distances on the order of hundreds of miles, it is feasible to use a carrier vehicle. In 2002, this concept was demonstrated at Edwards Air Force Base using the Predator (see Section 2.1.2) as the carrier vehicle [67]. During the demo a 57 pound mini-UAV was released at an altitude of 10,000 feet, performed a 25-minute preprogrammed mission, and then landed.

The state of the art for UAVs is moving toward MAVs where the size is smaller and the production cost cheaper so that these miniature UAVs can be used in hostile environments without concern for loss of life. While this very small UAV is not yet a reality, research is underway toward producing these MAVs of the future [70][93][77]. Having such versatile assets on hand during the battle at the front lines is exactly what is needed for the next century battlefield. Assuming that MAVs are readily available sometime in the future, it is expected that these devices also have sensors by which they monitor their environment. Considering that there are hundreds of these sensor-laden MAVs working toward a common goal, another area of research becomes relevant: distributed sensor processing.

2.2 Distributed Sensor Processing

2.2.1 Background Development. Sensor processing consists of four stages: acquisition, processing, integration, and analysis [94]. The first activity, acquisition, gathers data from the battlefield environment. This gathering can occur both passively and actively based on the type of sensor. The data gathered is composed of two parts: the useful part, or the signal, and the noise or error, which is the other part. Nonetheless, the data at this point is called raw data and must be processed in order to provide useful information. That is what begins to occur in the processing stage. It is the local computing that occurs on board with the sensor hardware. It is also called preprocessing for more complex systems. The third stage of activity is extremely important for distributed sensor systems: integration. This is also known as fusion and is also one of the most difficult activities in sensor processing. Fusion algorithms differ based on the kinds

of data that are being integrated. Competitive algorithms are used when redundant data needs fused. Cooperative algorithms fuse data that is non overlapping but partial while complementary algorithms fuse overlapping and partial data from each distributed sensor. Once the data has been fused, the integrated product is then analyzed which ends with a decision. A typical decision to be made for a military swarm reconnaissance network of sensors is how many targets are present?

Distributed sensor processing can occur on a variety of hardware configurations with many applications. The stages of sensor processing within a distributed environment occur in a distributed fashion at each sensor node or on an entirely separate compute node or downloaded to an off site station for post processing. Because this research places the sensors on a UAV platform it is necessary to consider wireless networking. The wireless network is the communication mechanism through which a UAV distributed sensor network performs meta-level sensor processing. Also relevant to this research are the movement patterns of the UAVs which directly affect the network topology. Because of the highly mobile nature of coordinated UAVs, the network must be able to react acceptably to the changes in network topology. Thus many designs exist for a distributed sensor processing system in which the level that data is processed, integrated, and analyzed can range from each local node doing its own individual analysis to a meta-level aggregate form of analysis. All such designs come from a combination of one or more of the following: centralized control and computing, centralized control and decentralized computing, and decentralized control and computing.

The remainder of this section presents insight into the subjects of distributed sensor networks and ad-hoc wireless reconfigurable networks.

2.2.2 Distributed Sensor Network Development. Distributed sensor networks (DSNs) have recently emerged as an important research subject [54]. The main goal of distributed sensor networks is to make decisions or gain knowledge based on the available information that is distributed across the sensor inputs. Applications of DSNs include both civilian and military tasks such as environment monitoring, scene reconstruction, motion tracking, motion detection, battlefield surveillance, remote sensing, and global awareness. Distributed sensor networks are not general-purpose communication networks, rather they are task-specific networks with a range of applications based on on-board sensors.

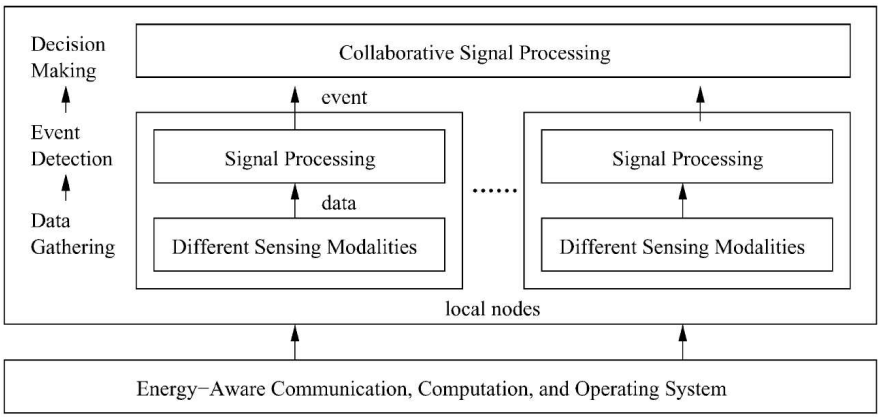


Figure 4 Block diagram of a DSN from functionality point of view

The authors of [83] present the diagram shown in Figure 4 to portray the different components in a DSN from the functionality point of view. It is a good frame of reference to gain an understanding. At the lowest level, individual sensor nodes collect data from different sensing modalities (i.e. modes). Initial data processing is carried out at the local node to generate a local event detection result. These intermediate results are then integrated at an upper processing center to derive knowledge and help making decisions.

While the concept is simply stated there are many unresolved research issues associated with the concept of DSNs all pertaining to the sensor fusion that occurs. The authors of [83] summarize them in question form: what to fuse? where to fuse? and how to fuse? With the size of sensors getting smaller and the price getting cheaper, more sensors can be developed to achieve quality through quantity. On the other hand, sensors typically communicate through wireless networks where the network bandwidth is much lower than for wired communication. These issues bring new challenges to the design of DSNs: first, data volumes being integrated are much larger; second, the communication bandwidth for wireless network is much lower; third, the power resource on each sensor is quite limited; fourth, the environment is more unreliable, causing unreliable network connection and increasing the likelihood of input data to be faulty.

2.2.3 Wireless Ad-hoc Networking Development. Advances in sensor technology and wireless communication have made ad-hoc sensor networks a reality. Traditional wired networks have fixed paths and a relatively static structure. On the other hand, an ad-hoc network has a

Table 1 Open Systems Interconnect Seven Layer Protocols

Layer	Description
Application	Protocols keep track of which application to send the message to. Defines the interface to a service. Provides special network access to applications.
Presentation	Protocols at this level transmit data in a network representation that is independent of the individual characteristics of each system (i.e. Operating System differences). Compression and Encryption occur here.
Session	Reliability detection and recovery occur at this level. This layer maintains the connection.
Transport	Messages (rather than packets) are handled from this level up to the application layer. Messages are addressed through communication ports attached to processes. Ensure that data is delivered at this layer.
Network	Transfers data packets between computers in a specific network. This is where data is directed to an address.
Data link	Responsible for transmission of packets between nodes that are directly connected by a physical link.
Physical	The hardware and electronics that drive the network. It transmits sequences of binary data by analogue signals.

topology that is dynamically changing. This results in a short-lived network being set up for the current communication need.

The topic of networking cannot be mentioned without making reference to the Open Systems Interconnect (OSI) seven layer reference model adopted by the International Standards Organization [79]. In order for any two electronic devices to communicate, they must speak a common language. This common dialog is guided by something called protocols for computing systems. The OSI specifies seven layers of protocols that enable multi-computer communication compatibility– see Table 1. The lower two layers are implemented in hardware and software, while the upper five layers are implemented in software. Each layer interacts with the layers adjoining it. It is not necessary (nor common practice) that every protocol layer be used for communication so that for a network of UAVs it is expected that only a small subset of the OSI reference model protocols are used. This simplification reduces the consumed bandwidth and power requirements at each node.

'Wireless' and 'Reconfigurable' imply several important factors that influence network structure. Wireless implies limited resources such as low power, limited bandwidth, low cost, and short distances. Reconfigurable likewise implies following connotations: mobile, intelligent, self-configuring, self-organizing, and unpredictable. With this sphere of influence, designing appro-

priate routing protocols for the wireless reconfigurable network is demanding. *Scalable* routing protocols is also of interest when considering a swarm of distributed sensors, because the number of nodes can range anywhere from the hundreds to thousands to millions depending on the mission.

Several routing protocols exist for ad-hoc networks and have been grouped into three broad categories by [48]: flat routing schemes including proactive and reactive, hierarchical routing, and geographic position assisted routing. Flat routing approaches adopt a flat addressing scheme where each node participating plays an equal role. Proactive flat routing is table-driven meaning that each node includes a lookup table of locations of all other nodes. Reactive flat routing, also called on-demand routing is a new routing philosophy in the ad hoc arena which does not store routing activities or information if there is no communication. Hierarchical, in contrast, usually assigns different roles to network nodes. Some protocols require a hierarchical addressing system. Routing with geographic positioning information requires each node to be equipped with the Global Positioning System.

2.2.3.1 Directed Diffusion Protocol. The preferred protocol of interest when applied to sensor networks is Directed Diffusion [49][58][55]–an on-demand routing protocol with little overhead resulting in energy savings through selecting empirically good paths by caching and processing data in-network. This data-centric protocol specifies that nodes are not addressed by IP addresses, but by the data they generate. All nodes in a directed diffusion network are application-aware. A node requests data by sending interests for named data. For example, an interest for a micro-UAV swarm reconnaissance mission might search out moving targets with a particular radius. Data from all nodes matching the reconnaissance information is sent to the requesting node. All communication in diffusion is neighbor-to-neighbor, unlike the end-to-end communication of traditional networks. The path on which this information travels is determined by interest propagation. Intermediate nodes cache, transform data, and direct interests based on previously cached information.

An example task described by attribute-value pairs might be:

```
{type = mobile vehicle,  //detect vehicle location
  interval = 3ms,        //send back events every 3ms
  duration = 10sec,      //for the next 10 seconds
```

rect = -10, 10, 20, 40 } //from sensors within rectangle

All sensors that pickup data within these constraints report that information back to the requesting node–which is a neighbor of the current sensor node. Data propagation and reinforcement is achieved through a gradient which is described as the natural flow of data to and from an interest event, i.e. vehicle movement. Since requests are sent by neighbors and then those neighbors to all their neighbors until the network is exhausted, then a gradient forms along the paths to all interests, i.e. moving vehicles, by virtue of the fact that those sensor nodes that do not perceive the road or paths where vehicles are traveling will not respond with any events. These weak interactions are overtaken by those sensors that do have events flowing regularly back to the request nodes such that a natural data-centric gradient is formed. The request nodes tweak this gradient through increasing or reducing the request interval and duration attribute-value pairs or some other customized criterion. Eventually, the event requests to sensors outside of the interest area expire and thus are removed so that only the empirical path to the event is reinforced by future queries.

Of course, several assumptions of the attribute-value information are made in support of the aforementioned mobile vehicle example. The design space is considerably large for directed diffusion and can be optimized for a given set of priorities. Robust data delivery, maximum sensor coverage, selective quality, and multi-path delivery with probabilistic forwarding are just a few of the possibilities discussed in [49]. Preliminary evaluation of this protocol revealed that directed diffusion has significant potential energy efficiency even with relatively unoptimized path selection. Also, it is stable under uniform and random sensor node failures.

2.2.4 Distributed Processing Development. Distributed sensor processing cannot be discussed outside the realm of the larger topic of distributed processing or parallel computing. Parallel computing takes advantage of High Performance Computing (HPC) resources through the use of distributing processor intensive pieces of a program across multiple processors. While a micro-UAV swarm will not have as much computing power as a HPC resource, it can still utilize the communication ideas and concepts present in a parallel computing environment.

Parallel algorithm design and data decomposition strategies can be applied to increase efficiency and provide effectiveness that could not otherwise be achieved. For example, using a single UAV platform such as the Predator to characterize an enemy's battlefront presentation would re-

quire multiple vantage points from which various resources could be detected positionally and chemically. Using a parallel strategy (i.e. micro-UAV swarm) to accomplish this same goal would increase probability of success through sheer numbers as well as distributing the sensing task among the participating nodes. Decomposing a battlefield into either a rectangular grid coordinate system or an application based organization allows for the swarms to divide and conquer the region of interest in an efficient manner.

Another distributed processing technique that can implemented in a distributed sensor network is dynamic load balancing. For parallel processing, when a node runs out of work it queries neighboring processors or a master node for more work thus accomplishing more work in less time while maximizing use of resources. Similarly, when a swarm of sensors detects a region of high density it can call on under-utilized nodes to keep the system from being overwhelmed.

2.3 Swarming Development

Distributed sensors provide the data collection mechanisms for the each one of the coordinated UAVs but they rely on emergent behavior algorithms to establish their movement. The paragraphs below give insight into how recent developments in swarm movement algorithms apply to this research.

Swarms were noticed in nature when biologists recognized a common simple behavior pattern in schooling fish, flocking birds, and bee swarms to name a few. Thus, a swarm is a collection of individuals working together. What is fascinating about swarms is that no central control figure dictates the behavior of the whole group, rather each individual reacts and responds to every other individual within his sphere of influence, resulting in a global behavior pattern such as a flock of birds. The result is an autonomous system–a key aspect in a micro-UAV distributed sensor environment.

2.3.1 Emergent Behavior. Emergent behavior is derived from the word emerge which means "to come forth from, to rise out of" according to Webster's Revised Unabridged Dictionary (1913). What is coming out or emerging is a global behavior. No single individual is causing the entire group to act in a certain manner, rather each one is directly influencing the others around him. It is in that manner that a global behavior *emerges* from the result of the smaller interactions

Table 2 Dudek's Swarm Classification Categories

Category	**Description**
Collective Size	The number of autonomous agents in the collective
Communication Range	The maximum distance between two elements of the collective such that communication is still possible
Communication Topology	Of the robots within the communication range, those which can be communicated with
Communication Bandwidth	How many information elements of the collective can transmit to each other
Collective Reconfigurability	The rate at which the organization of the collective can be modified
Processing Ability	The computational model utilized by individual elements of the collective
Collective Composition	Are the elements of the collective homogeneous or heterogeneous

that occur between the individual members. Put a different way, each member is following a simple rule set, from which a larger objective is accomplished. Thus a swarm's behavior is emergent.

Swarming is an emergent behavior of simple autonomous individuals according to [22]. Simply stated, using swarms is the same as "getting a bunch of small cheap dump things to do the same job as an expensive smart thing" [22]. Formally, it is a collection of autonomous individuals relying on local sensing and reactive behaviors interacting such that a global behavior emerges from the interactions [22]. No matter the definition all contain the idea that multiple entities work toward the same end result. All members have locally controlled behavior constrained by simple rules. All have predesigned reactive behaviors.

2.3.2 Swarm Taxonomy . Swarms in nature are best characterized by their behavior. When attempting to mimic the behaviors found in nature, a swarm model is not limited to simply the movement of the individual members of the swarm but all the faculties of the swarm. For example, a classification of swarm behavior is shown from [32] in Table 2. In it, Dudek indicates that such items as configuration of the communications topology plays a role in the classifying a swarm. Kadrovach introduces a swarm taxonomy as shown in Figure 5 that uses a three tier continuum: scale, coupling, behavior [53]. The examples he gives include a single large school of fish as a {global, ordered, loose} swarm, and a colony of ants foraging in widely scattered groups as a

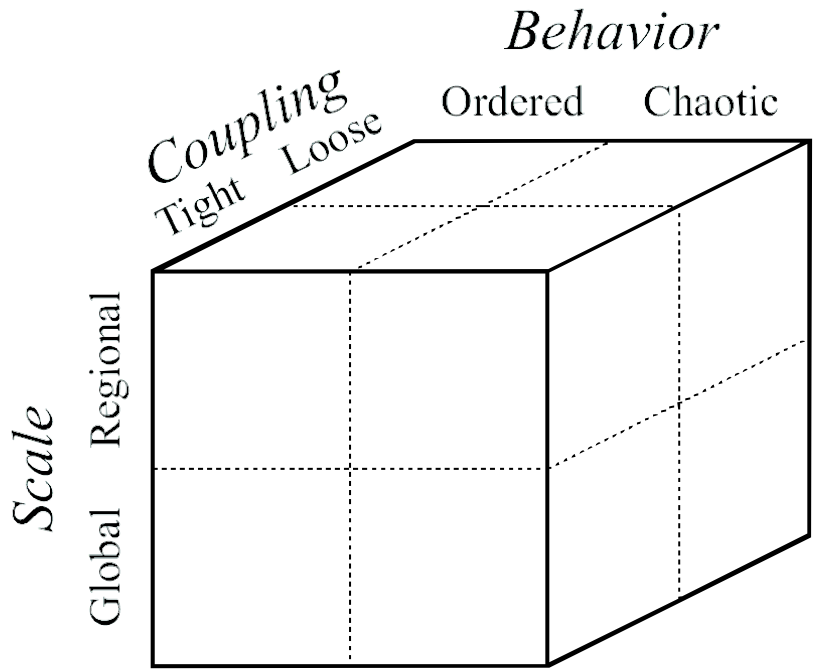


Figure 5 Kadrovach's Swarm Classification

{regional, chaotic, tight} swarm. Both of these classification methods are helpful in understanding the characteristics in nature and in man-made swarm models that represent "swarming."

2.3.3 Swarm Coherence. Swarm coherence is a key concept discussed in [53]. Somewhat related to swarm taxonomies is the idea that certain aspects of a swarm's *movement* are identified relative to coherence. Coherence in this context implies that the global direction of the swarm is aligned. While each member of the swarm may not be heading in exactly the same direction, the difference of each member from the overall direction of the swarm is within a specified tolerance. Thus, a swarm that is moving in relatively the same direction is said to be *coherent*, while a swarm that has members with large angular differences from the global direction is said to be *incoherent*. Kadrovach uses this concept in identifying two characteristic behaviors that appear in swarming applications.

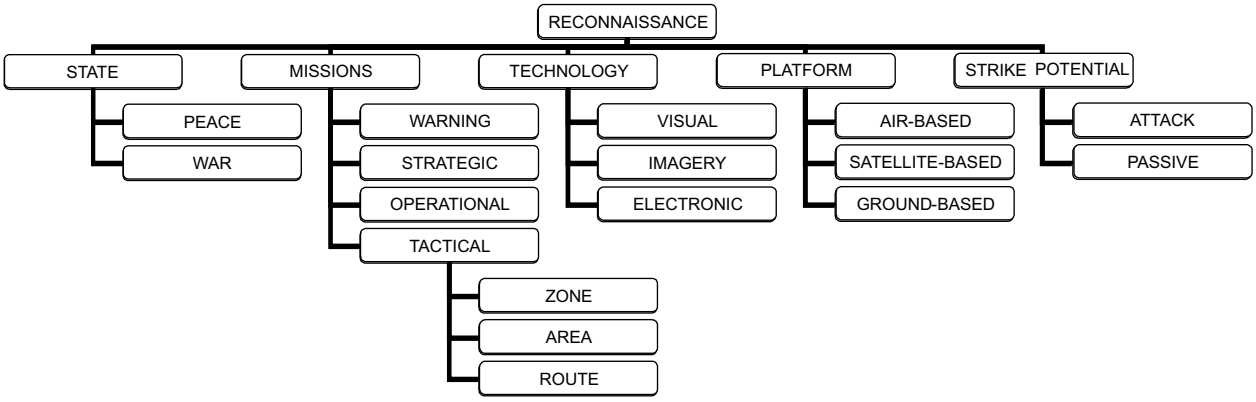


Figure 6 Reconnaissance Taxonomy

2.4 Reconnaissance

Using the emergent behavior and coherence of sensor swarms offers a unique approach to performing a reconnaissance operation. Reconnaissance appears in the broad topic of the C4I framework–command, control, communications, computers, and intelligence. Often it is grouped with the similar information gathering operations of surveillance and target acquisition. While this context brings additional insight and complexity, this work only considers one component in that framework: *reconnaissance* operations.

In general the definition in [7] for reconnaissance is the acquisition of information by employing visual observation and/or sensors in vehicles. Any information gathering act during war or peace time efforts by any military has many forms and methods of employment. Depending on when this mission is inserted into the battlefield and for what purpose, its success can spread an entire spectrum. The objectives of a successful reconnaissance mission vary according to the purpose. Those characteristics that are indicators for the purpose of a reconnaissance operation are presented in a meta-level taxonomy in Figure 6. The taxonomy shows that there are four major categories that spread across a reconnaissance operation: STATE, MISSIONS, TECHNOLOGY, PLATFORM, and STRIKE. Those major categories are then further divided to reveal the various characteristics that describe a continuum of unique reconnaissance operations. For example, an electronic technology reconnaissance war-time operational mission includes radio-frequency (RF) signal collection hardware from a ground-based station. Such an operation is synonymous with a ground station radar detecting RF emissions during a regional search across a no-fly zone. Another

example operation is a helicopter patrol performing a reconnaissance route search operation using visual (human eye) technology to find downed airmen in a heavily wooded enemy territory.

The STATE of the nation is a major division among reconnaissance operations. During war-time the tempo, environment, efficiency, and integration of the reconnaissance operation are much more critical than during a peace-time operation. In addition, terrorism can also be included as a sub category of either peace or war time operations imposing certain limits and restrictions on the operation.

The MISSIONS are spread across 4 categories: warning, strategic, operational, or tactical [5, 7, 30]. The Marines provide a meta-level description of an air-based reconnaissance approach to three of these categories:

- *Strategic* air reconnaissance provides intelligence information required to form plans and policies at national and international levels ... Locate threat centers of gravity and strategic targets.
- Air reconnaissance performed at the *operational* level provides information that is crucial to the planning and execution of theater-wide operations ... Help define the critical vulnerabilities of a threat's national structure and military capabilities.
- At the *tactical* level, air reconnaissance operations support ... mission planning, targeting, combat assessment, threat assessment, target imagery, observation of ground battle areas, targets, or sections of airspace. [7]

The fourth is mentioned in [30] in a perspective that discusses leveraging reconnaissance for future military strategy. Missions during peacetime provide indications and *warning* of all adversary deployment activities. The warning missions are further described in the U.S. doctrine for reconnaissance for joint operations:

Indications and *Warning*:

- ...provide information necessary to assess forces and installations that threaten the United States and its allies
- ...provide timely indications and warnings of a threat or impending attack

- ...provide information to assess force strength and deployment, defensive and offensive capabilities, and other factors that may affect US and/or allied military plans and operations. [5]

Similar discussion of reconnaissance missions from a ground-based perspective is found in [6]. In addition to the various missions of reconnaissance an associated risk rating presents a dichotomy between close and deep reconnaissance operations as the authors of [101] discuss. Deep reconnaissance identifies high value targets, gathers information on the enemy's capabilities, intent, and will all the while not posing a threat to the operator's life. Close reconnaissance on the other hand identifies specific enemies executed in a high threat environment where loss of resources is probable. In an effort to encompass all aspects of the mission categories, this research focuses on close reconnaissance tactical level missions because they can be the most complicated and demanding missions. The remaining missions can then be examined as a subset of the tactical that are performed with similar approaches but with less severity and intensity.

The TECHNOLOGY employed in a reconnaissance operation can be one of many forms. If using humans as the main information gathering databases, night vision goggles and other human-aided vision devices significantly enhances the *visual* ability. *Imagery* reconnaissance technology is widely used today for producing mass pictures of the entire globe and real-time pictures of hot interest areas around the world. *Electronic* technology includes the invisible portions of the electromagnetic spectrum that span the globe such as radio emissions, radar emissions, and infrared signatures. To reduce the complexity of the application, this research looks only at imagery type sensor technology employed during an operation.

PLATFORMs used to perform reconnaissance include all forms operating in all atmospheres to include space-based satellites. Ground based reconnaissance is a primary function of the Army's cavalry and includes not only ground troops but also ground vehicles [6]. Air based reconnaissance applies to all military branches: Army, Navy, Marines, and Air Force [5, 7]. The swarm approach used in this research implies the platform is performing air-based reconnaissance.

Having the ability to STRIKE during a reconnaissance operation presents another wide variety of applications. Some authors have included an offensive strike as part of their definition of reconnaissance [6, 101]. Striking operations include screen, cover, guard, attack, defend, counter-

recon, and impede. All of these require the platform to positively interact with the enemy. Without striking capability a reconnaissance operation is purely passive–unless destroyed by enemy forces. Research is best done by first laying a foundation and then increasing the number of parameters thus this work focuses on a simple non-striking reconnaissance force.

Some prerequisites are necessary for effective air reconnaissance which rely on certain characteristics about the capability of the vehicles performing the operation. Ideally, the basic prerequisites of air superiority, suppression of enemy air defenses, cooperative weather, capable platforms and sensors, and flexible control are necessary to perform a mission without losing effectiveness or being exposed to unnecessary risks for an air-based reconnaissance mission. For the purposes of this study, it is assumed the mission being performed is during an initial wartime scenario where air dominance has not been established and enemy air defenses are still operational. The element of cooperative weather is always a bonus but can quickly complicate things, therefore the weather variable is assumed acceptable. The capable platforms and sensors along with their control are discussed next.

Current researchers have explored future war-fighting concepts that involve reconnaissance members in a strike package, terrain features, numbers of forces, and simulated engagements [42]. They simulated a red force against blue force model in a two-dimensional varied terrain play box with each force containing some mix of high lethality, low protection strike agents and reconnaissance agents. The following parameters were among those that were varied: number of reconnaissance assets traded one for one with strike assets (Red <-> Blue), combat threshold, and weapon range. This results in statistics that tell which force survived and how much damage was sustained while derived metrics indicate the value added for other various parameters. One sample observation from the paper is that investing in reconnaissance or additional strike assets can lead to the blue force being more successful as terrain complexity increases. More generally, it was found that the modeling of surveillance and intelligence was very difficult as was generating variable interaction between force types. While this paper does provide insight into one method of measuring reconnaissance success it does not provide the needed reconnaissance model to measure the ability of autonomous swarming UAVs to perform reconnaissance.

2.5 Parallel Discrete Event Simulation

With this section, the focus of the chapter shifts from the real-world objects (swarms of sensor UAVs) to simulation of those objects. The shift is necessary because it is not reasonable to research the above ideas in the physical world given the limited resources available for this effort. Instead, a simulation of the swarm of sensor UAVs is to be designed and tested. This section presents pertinent developments in parallel simulation to equip the reader with an understanding of its fundamental concepts.

2.5.1 Background Development. A simulation is "the imitation of the operation of a real-world process or system over time" [12]. It involves exploring the behavior of that system by developing a simulation model. Simulation commonly falls into one of two categories: event-based or time-based. Discrete event simulation simply means that only particular events of interest are included in the simulation rather than mimicking every single event that occurs in the corresponding continuous real-world process. Said another way, it is the modeling of systems in which the state variable changes only at a discrete set of points in time [12]. Discrete event processing is contrasted with a time-stepped or synchronous simulation which normally updates the entire state of the modeled system at regular constant time intervals. The advantage of event-based simulation is that updates can be scheduled to occur only when necessary, unlike the time-stepped simulation where the progress of the simulation is limited by the incremental time step whether the state of the model changes or not for that particular instant.

Simulation requires that the system be decomposable into objects that make up the simulation. Each of the objects must include the attributes necessary to imitate what occurs in the real-world system. Common to all simulations are several terms that aid in this system decomposition. Table 16 presents the terms along with their definitions as discussed in [12].

A parallel simulation occurs with multiple processes rather than one sequential process. Each logical process can reside on one machine or they can be distributed across a network of machines. Parallel Discrete Event Simulations (PDES) therefore allow for simultaneous execution of events that are unrelated and thus increasing the overall efficiency of the simulation. The PDES model presented next explains this advantage in mathematical terms.

Table 3 Simulation Components of a System

Term	Definition
entity	object of interest in the system; it requires explicit representation in the model
attribute	property of an entity
activity	a time period of specified length with a known start time
sytem state	the collection of variables necessary to describe the system at any time, relative to the objectives of the study
event	instantaneous occurrence that may change the state of the system
event notice	record of an event to occur at the current or some future time; includes any associated data necessary to execute the event
event list	list of event notices for future events ordered by time of occurrence
endogenous	activities and events occurring within a system
exogenous	activities and events in the environment that affect the system
system	collection of entities that interact together over time to accomplish one or more goals
model	abstract representation of a system; containing stuctural, logical, or mathematical relationships which describe the system in terms of state, entities, sets, processes, events, activities, and delays
list (queue or chain)	collection of associated entities, ordered in some logical fashion
delay	duration of time of unspecified indefinite length, which is not known until it ends
clock	variable representing simulated time

2.5.2 PDES Mathematical Model Representation . Common to all parallel simulation strategies is their aim to divide a global simulation task into a set of communicating logical processes (LPs), trying to exploit the parallelism inherent among a distributed simulation architecture. In [34], a basic architecture for a PDES is developed. It is reproduced here.

"A logical process simulation (LP simulation) can be viewed as the cooperation of an arrangement of interacting LPs, each of them simulating a subspace of the space-time which is called an event structure region. Generally a region is represented by the set of all events in a subepoch of the simulation time or the set of all events in a certain subspace of the simulation space. The basic architecture of an LP simulation can be viewed as in Figure 7.

- A *set of LPs* is devised to execute event occurrences synchronously or asynchronously in parallel.

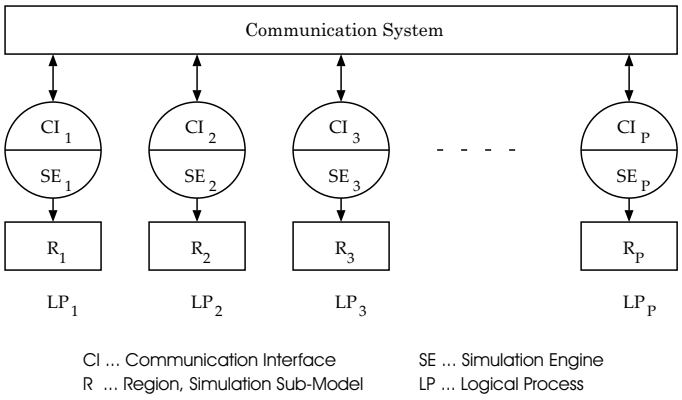


Figure 7 Architecture of a Logical Process Simulation [34]

- A *communication system* (CS) provides the possibility for LPs to exchange local data but also to synchronize local activities.
- Every LP_i has assigned a *region* R_i as part of the simulation model upon which a simulation engine SE_i operating in event driven mode executes *local* (and generates *remote*) event occurrences, thus progressing a *local clock* (local virtual time, LVT)
- Each LP_i (SE_i) has access only to a statically partitioned *subset of the state variables* $S_i \subset S$, disjoint to state variables assigned to other LPs.
- Two kinds of events are processed in each LP_i: *internal events* which have causal impact only to $S_i \subset S$, and *external events* also affect $S_j \subset S\,(i \neq j)$the local states of other LPs.
- A *communication interface*, CI_i attached to the SE takes care for the propagation of effects causal to events to be simulated by remote LPs, and the proper inclusion of causal effects to the local simulation as produced by remote LPs. The main mechanism for this is the sending, receiving and processing of event messages piggybacked with copies of the senders LVT at the sending instant. " [34]

This research focuses on *asynchronous* parallel simulation. Thus for increased efficiency to be realized it is necessary that events occur at different simulation times which do not affect one another. Concurrent processing of those events thus effectively accelerates sequential simulation execution time. The critical concept in order for this to occur then is the preservation of the causality of the system being simulated. That is every asynchronous LP does not produce causality

errors. This is studied in detail in [73], in which it is shown that no causality error can ever occur in an asynchronous LP simulation if and only if every LP adheres to processing events in nondecreasing time stamp order only–called *local causality constraint* in [38]. It is shown in [33] that the potential performance improvement of an asynchronous discrete event simulation strategy over the time-stepped variant is at most $O(log\, P)$ where P is the number of logical processes executing concurrently on independent processors.

2.5.3 PDES Operational Architecture Development. Several operation configurations exist for PDES computing including Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Multiple Data (MIMD), and Single Program Multiple Data (SPMD) [44]. All of these modes perform operations on data. How that operation occurs and on which data distinguishes each mode. For example, in a SIMD environment a set of processors performs identical operations on different data in lock step. Each processor possesses its own local memory for private data and programs, and executes an instruction stream controlled by a central unit. The SIMD controller forces synchronism among the independent computations.

The desired mode of operation for a PDES is the MIMD. In this mode, multiple processors execute independent instruction sets on different data. According to [34], "A collection of processes assigned to processors operate asynchronously in parallel, usually employing message passing as a means of communication. In addition to the data exchange, synchronization must occur through the communications backplane. The generality of the MIMD model adds another difficulty to the design, implementation and execution of parallel simulations, namely the necessity of an explicit encoding of a synchronization strategy." For this research, that extra design difficulty pertains to the synchronization of UAV position update events and is discussed in the next chapter. Using the MIMD operation model provides the simulation with an implicit method of simultaneous event execution because of the independence of each processor's instruction set and data stream.

2.5.4 High Performance Computing Development. The MIMD architecture typically resides on a High Performance Computing (HPC) system. One HPC system, called Beowulf, is defined below:

> a kind of high-performance massively parallel computer built primarily out of commodity hardware components, running a free-software operating system like Linux or FreeBSD, interconnected by a private high-speed network. It consists of a cluster of PCs or workstations dedicated to running high-performance computing tasks. The nodes in the cluster don't sit on people's desks; they are dedicated to running cluster jobs. It is usually connected to the outside world through only a single node.[15]

The Beowulf clusters available at the Air Force Institute of Technology (AFIT) include Aspen, Pile of PCs, Poly, and Myrinet. Aspen, the Pile of PCs, and Poly all use a standard fast Ethernet network with a data transfer rate of 100 M$\frac{bits}{sec}$=$100 \times 2^{17} \frac{bytes}{sec}$. Myrinet is an American National Standard – ANSI/VITA 26-1998 [76]. It has a data transfer rate when in single duplex mode of $2G\frac{bits}{sec} = 2 \times 2^{27} \frac{bytes}{sec}$. Making a program compatible with an HPC platform involves using the Message Passing Interface (MPI) language constructs in the high level programming language such as Java and C++. The MPI constructs supported by the systems at AFIT include MPICH and GM-MPICH which are both free software distributions. In addition to these specialized high speed message passing constructs the standard TCP protocols are supported.

2.6 Summary

This chapter explored current research developments in the many subjects associated with a parallel swarm simulation of a reconnaissance application. A thorough development of the UAV as a weapon system leads into current research efforts toward Micro UAV technology. Next, the challenges of distributed sensor networks highlights the importance of a well defined sensor data processing/fusing strategy. The role of communication in that strategy then outlines the challenges associated with a mobile ad-hoc networking but it is accompanied with an applicable solution–the directed diffusion routing protocol. Swarming developments include three inter-related concepts of emergent behavior, classification of behavior, and coherence. Next, the key developments of reconnaissance prepare the reader for scenario design. Finally the simulation layer–Parallel Discrete Event Simulation–introduces a mathematical representation to facilitate understanding key parallel computing concepts. The next chapter begins with the high level design of a system that is founded on the ideas presented in this chapter.

3. High Level Design

3.1 Introduction

This chapter discusses the higher level design aspects involved in designing a parallel swarm reconnaissance application. First the reconnaissance mission is defined, next the design principles for parallel discrete event simulations are presented. Then model fidelity design issues are presented along with particular algorithm selection for the major models in this research effort. The design decisions result in justification of the level of simulation fidelity in each respect. This chapter concludes with two high level design areas that discuss software engineering principles and parallel computing design objectives as applied to a parallel swarm simulation.

3.2 Reconnaissance Mission Design

This research focuses on the tactical level application of reconnaissance that supports both commanders in the battle space and soldiers on the front lines of battle with an air-based platform equipped with imagery sensors an autonomous mobility. Prior to making any design decisions for models, implementations, or processes it is critical for one to understand the objective those tools are to accomplish. That is where reconnaissance mission design comes in. Designing a realistic mission involves defining the environmental influences, expected limitations, scope of reconnaissance, and associated mission variables. This section presents those variables.

3.2.1 Scenario Parameters. To accommodate the spectrum of reconnaissance missions three distinct scenarios of varying difficulty are chosen . The first is only applicable in a pedagogical sense and thus is designed for understanding rather than representation. The second scenario has a lower level of complexity than a typical mission in the real world. The final scenario is the closest representation of the real world.

Generically, all reconnaissance operations are providing situational awareness of a battle front. The Department of the Army's field manual on the cavalry reconnaissance troop describes this situational awareness as a clear picture of the battlefield framework [6]. This framework is described as the conditions of mission, enemy, terrain, troops, and time (METT-T) and includes the location of friendly troops, the range of direct-fire weapons, observation, sensors, and the terrain

Table 4 Reconnaissance Scenarios

Scenario Name	Terrain	E	Enemy poise	T	TM	Swarm size	World dimensions	Operations
Pedagogical	desert	5	-	5	S	10	1000x1000	zone, area, route
Passive	plains	8	monitoring	15	S	100	3000x3000	zone, area, bda
Active	jungle	20	pursuing	50	M	1000	5000x5000	zone, area, bda

KEY[E number of enemy targets; T total number of enemies (including threats); TM target mobility; S Stationary; M Moving]

on which they are applied. These are the kinds of scenario parameters that are used to design a mission.

Table 4 outlines the specific parameters defining three different scenarios that are used in this research. The first is a pedagogical example that is useful in explaining the logic behind the simulation models and how the algorithms work. The second is a more challenging setup that demands more of the model and the third an even larger challenge. The parameters that distinguish each are shown across the top row in the table. The terrain aspect of the simulation strives to add realism to the simulation by demonstrating the capability of the swarm across several environments. The desert environment does not provide many hiding places unlike a jungle or cavern environment might. The number of enemies includes both the "threats" and targets. It is assumed for this research that a "threat" is not a target but rather is either a ground radar platforms or future anti-swarm systems that seeks and destroy swarms. A target is any enemy person/structure/vehicle that is defined by the commander. Thus the total number of enemies includes both "threats" and "targets" so that the $\#targets = total_{enemies} - \#threats$. Enemy poise indicates the tempo of the enemy. A pursuing enemy does not only have all of its threat warning systems active (threats) but it also has made a concerted effort to conceal any high value targets. A monitoring poise is less active and does not include all available active threats or extra efforts to conceal high value targets. Target mobility is defined as "S" for stationary or non-moving and "M" for mobile or moving targets. Swarm size is simply the number of UAVs that make up a swarm. The world dimensions is the area in which the swarm, threats, and targets all reside. This implies there are boundaries. The operations are reconnaissance modes as discussed in 2.4. Zone mode follows a restricted subset of the world with the intention of finding potential intruders from a vulnerable exposure, while area

mode simply searches an entire defined area. Route mode follows a particular military route ahead of an envoy alerting the commander of any potential ambushes. Battle damage assessment (BDA) is simply recording pictures of a target that has been lethally engaged, thus the objective is to locate and report on a variety of positions within the world.

3.2.2 Measures of Effectiveness (MOE). Effectiveness usually means whether or not the mission accomplished the desired task. Thus the mission task must be clearly defined. For the purposes of this research the effectiveness of a reconnaissance mission is defined as a measure of the percentage of the targets being detected. The success and failure criteria follow.

- MOE: Target Identification
  - Success: 90% of the targets are reported.
  - Failure: < 90% of the targets are reported.

A reported target is one that has been identified by the reconnaissance swarm and has been reported to the command post. Depending on the application, the command post can be a soldier that is scouting out the front line operations or it can be the ground commander residing in a hardened remote control center. The 90% figure comes from the realism that is part of the world that we live in. Often, during wartime the risk of losing human life plays a key role in determining the measured success of an operation. As such, this research is focused toward a medium level risk situation where the detection of 90% of the targets incurs acceptable risk for continued operations. This number can be tailored and applied to future mission requirements.

Another potential MOE for a reconnaissance could be target destruction, however since the vehicles used in this research are not capable of delivering munitions this MOE does not apply.

3.2.3 Measures of Performance (MOP). Several principles introduced in [7] ensure valued information is the product of a reconnaissance operation rather than questionable sensor data. They include accuracy, balance, relevance, and timeliness. Reconnaissance information must be accurate and reliable. Accuracy consists not just of reporting objects in the battle space, but rather providing as much information as possible about each object, chief of which is the object's position. Geo-position accuracy is a crucial requirement when employing global positioning system

guided munitions. Accuracy also indicates the level of resolution of a target, such as recognizing the difference between a real tank and a cardboard mock up shown from an image sensor. Balance is necessary when the reconnaissance operation is exposed to a threat or is gaining a higher resolution image. Increased accuracy requires extended observation and analysis times but at the same time the risk of the mission increases with the presence of threats or delays the delivery of the intelligence to the commander. Balance allows some acceptable risk while maintaining an effective response time. Relevance pertains to the usability of the data. It must be relevant to the user and presented in a format that is useful for the decision making process. Timeliness encompasses the ideals of surprise, defense, initiative, an effective use of force. Data must be available in time to plan and execute operations. Without maintaining an element of surprise the enemy can react before an attack, without timely data a valid defense might be setup in the wrong position, without timely data the enemy will have the initial attack and effective use of his forces.

Thus performance of a reconnaissance mission can be measured by accuracy, balance, relevance, and timeliness. The latter two principles are related to the first, thus only 2 MOPS are used in this effort.

- MOP1: Accuracy: How well does the detected position reflect the target's actual position? This metric can be severely eroded when a target is moving, thus it is important to define accuracy in terms of time and space. In addition, the communication infrastructure can negatively impact accuracy of a reconnaissance mission because of inherent latencies and communication failures. Both the failures and delays can cause a target's position to become outdated. To maintain the scope of this research, only one aspect of accuracy is measured.
  - Positional Accuracy: Difference between actual position at time `t` and reported position at time `t`.
- MOP2: Balance: How much does the response time vary based on threat avoidance or communication delays. Response time is defined as the time it takes to deliver a target message to the command post once the swarm has identified it. When target interests are directed by the command post, the response time is the time it takes from the receipt of the defined interest to the reporting of data on that interest.

– Response Balance: Average response time without presence of threats as compared to the response time during a threat presence. Obviously, one would expect the response time to increase when the swarm is threatened.

While not all tactics of reconnaissance have been considered (i.e. recon-attack), a basic subset are present. Ideally a simulation of these scenarios allows for both the MOEs and MOPs to be evaluated against all variables from Table 4. Next, the possibilities in simulation design toward that ideal are given.

3.3 Parallel Discrete Event Simulation Design

3.3.1 Optimistic versus Conservative Schemes. Parallel discrete event simulations process events in a synchronous locked fashion meaning that one event that occurs earlier in simulation time precedes other events that occur later in simulation time. Often these simulation events have causal relationships that create dependencies among a chain of events. However, one can take advantage of events that are not related by processing them out of order thus potentially decreasing total simulation time. For example, if the next event in the queue to be processed does not occur for another 10 simulation seconds other events that are asynchronous can be safely processed either by a different processor in a parallel computing environment or by a separate process for a single processor system. Either way the lost CPU cycles can greatly improve the efficiency of the simulation. Processing all events in a synchronized fashion without any out-of-order event execution is called conservative processing while assuming all events are unrelated and processing them out-of-order most of the time is called optimistic processing.

3.3.1.1 Conservative. Originally developed by Chandy, Misra, and Bryant, (CMB) the conservative protocol preserves the causality of events across LPs by sending timestamped (external) event messages. The following formal notation is from [34]. A *conservative* logical process (LP^{cons}) is only allowed to process safe events. A safe event is one that is valid up to a local virtual time (LVT) for which the logical process (LP) has been guaranteed not to receive (external event) messages that are in the past according to the current LVT. This means a conservative approach follows the local causality constraint (lcc), see Section 2.5.2. All events must be processed in chronological order, which guarantees that the message stream produced by an $LP^{cons}$is in turn

in chronological order and a communication system preserving the order of messages sent from LP_i^{cons} to LP_j^{cons}(FIFO) is sufficient to guarantee that no out of order message can ever arrive in any LP_i^{cons}. A conservative LP simulation can thus be seen as a set of all LPs:

$$LP^{cons} = \bigcup_k LP_k^{cons} \tag{1}$$

together with a set of directed, reliable, FIFO communication channels:

$$CH = \bigcup_{k,i\,(k \neq i)} ch_{k,i} = (LP_k, LP_i) \tag{2}$$

that constitute the *Graph of Logical Processes*:

$$GLP^{cons} = (LP, CH) \tag{3}$$

Therefore a conservative approach strictly avoids the possibility of any causality error ever occurring. When a process contains no safe events it must block thus introducing the possibility of deadlock situations. The authors of [34, 38] include overviews of the various approaches to avoiding/detecting/recovering deadlock within a simulation.

3.3.1.2 Optimistic. Optimistic LP simulation approaches in contrast to conservative ones, do not strictly adhere to the *lcc*, but allow causality errors and provide a mechanism to resolve those *lcc* violations. Serious performance pitfalls of the conservative approach include blocking and safe-to-process determination. These pitfalls are addressed by allowing an optimistic LP to advance LVT as far into the simulated future as possible without guarantee that the set of generated events is chronologically consistent.

The well-known Time Warp algorithm pioneered by Jefferson and Sowizral [51] is one optimistic approach that deals with the problem of out-of-order execution. It employs rollback (in time) mechanisms to ensure proper synchronization. This requires a significant record of state history data which is a drawback to optimal processing. When an event is encountered that is scheduled

for a time that has already past, a series of anti-messages are sent to LPs as a request to annihilate the prematurely executed event prior that was computed based on causally erroneous state.

3.3.1.3 Comparison. A comparison between the two approaches does not result in an obvious solution. Ferscha presents an excellent side-by-side comparison showing the various influencing factors on each approach [34]. This comparison is shown in Table 5. This research intends to leverage the benefits of optimistic processing during the simulation event process cycle and thus prefers an optimistic time management scheme as available.

Table 5: Conservative vs Optimistic Strategies

Strategy	Conservative (CMB)	Optimistic (Time Warp)
Operational Principle	local causality constraint (lcc) violation is strictly avoided; only safe ("good") events are processed	lets lcc violation occur, but recovers when detected (immediately or in the future); processes "good" and "bad" events, eventually commits good ones, cancels bad ones
Synchronization	synchronization mechanism is processor blocking; as a consequence prone to deadlock situations (deadlock is a protocol intrinsic, not a resource contention problem); deadlock prevention protocols based on null messages are liable to sever communication overheads; deadlock detection and recovery protocols mostly rely on a centralized deadlock manager	synchronization mechanism is rollback (of simulated time); consequential remote annihilation mechanisms are liable to severe communication overheads; cascades of rollbacks that will eventually terminate can burden execution performance and embody utilization
Parallelism	model parallelism cannot be fully exploited; if causalities are probable but seldom, protocol behaves overly pessimistic	model parallelism is fully exploitable; if causalities are probable but frequent, the Time Warp can gain most of the time
Lookahead	necessary to make CMB operable, essential for performance	Time Warp does not rely on any model related lookahead information, but lookahead can be used to optimize the protocol
Balance	CMB performs well as long as all static channels are equally utilized; large dispersion of events in space and time is not bothersome	Time Warp performs well if average Local Virtual Time (LVT) progression is "balanced" among all LPs; space time dispersion of events can degrade performance
Global Virtual Time (GVT)	implicitly executes along the GVT bound; no explicit GVT computation required	relies on explicit GVT which is generally hard to compute; centralized GVT manager algorithms are liable to communication bottlenecks if no hardware support; distributed GVT algorithms impose high communication overhead and seem less effective

Table 5: *continued...*

Strategy	Conservative (CMB)	Optimistic (Time Warp)
States	conservative memory utilization copes with simulation models having "arbitrarily" large state spaces	performs best when state space and storage requirement per state is small
Memory	conservative memory consumption (as a consequence of the scheme)	aggressive memory consumption; state saving overhead; fossil collection requires efficient and frequent GVT computation to be effective; complex memory management schemes necessary to prevent memory exhaustion
Messages and Communication	timestamp order arrival of messages and event processing mandatory; strict separation of input channels required; static LP interconnection channel topology	messages can arrive out of chronological order, but must be executed in timestamp order; one single input queue; no static communication topology; no need to receive messages in sending order (FIFO), can thus be used on more general hardware platforms
Implementation	straightforward to implement; simple control and data structures	hard to implement and debug; simple data structures, but complex data manipulations and control structures; "tricky" implementations of control flow (interrupts) and memory organization essential; several performance influencing implementation optimizations possible
Performance	mainly relies on deadlock management strategy; computational and communication overhead per event is small on average; protocol in favor of "fine grain" simulation models; no general performance statement possible	mainly relies on excessive optimism control and strategy to manage memory consumption; computational an communication overhead per event is high on average; protocol in favor of "large grain" simulation models; no general performance statement possible

3.3.2 PDES Design Principles. To exploit the parallelism of a PDES application it is necessary to take note of certain design principles. When considering a parallel execution of a simulation model running on a cluster of processors there are three primary sources of overheads [10]:

1. Partitioning related overheads

2. Synchronization protocol overheads

3. Target architecture overheads

Partitioning related overheads are due to the model's decomposition and associated overhead contributions by load (im)balance, message communications, and related factors. Synchronization contributes overhead time by the specific algorithmic instances of managing the null messages and processor blocking for a conservative protocol and rollback and checkpoint overheads for an optimistic protocol. Target architecture overhead costs include the message latency and context switching overheads that are in many cases beyond the control of the user.

Bagrodia in [10] presents 10 pitfalls to avoid when designing PDES applications. Based on his discussion several guidelines are developed by the author during design of the parallel swarm simulation used in this research. Table 6 shows those guidelines.

Producing a high performance parallel simulation relies on models that are being simulated as shown in the next section.

3.4 Algorithm Models

What models are needed for an accurate simulation of a swarming reconnaissance mission? This section answers that question and provide details of the available alternatives.

3.4.1 Swarm Behavior Model. Several mathematical swarm models have been developed by researchers. A partial list includes particle swarm simulation [103], physical robots [64], minefield clearing [21], cooperative control of autonomous air vehicles [68][81], chemical cloud detection [56], and distributed sensor networks [54]. In addition to these educational researchers, the Air Force Research Laboratory Control Automation Area is currently interested in the feasibility and usefulness of swarms [22]. This research intends to use an existing model of acceptable fidelity.

An accurate model represents the real-world object at the highest level of fidelity. The corresponding real-world object being modeled are those evidences in nature as discussed in section

Table 6 PDES Design Guidelines

#	Pitfall	Guideline
1	Shared Variables	Eliminate global variables. Shared variables may be eliminated from a program by transforming it such that the variables duplicated across LPs or read and write operations implemented via messages or other mechanisms for interprocess communication.
2	Pointer Data Structures	Use pointers sparingly. Because there is no logical sharing of data between LPs located on separate processors data cannot be passed between LPs via pointers rather it must be passed by value.
3	Zero Delay Cycles	Always schedule events for a time greater than the current simulation time. A model with a zero delay cycle occurs when a sequence of messages exchanged have the same timestamp. This can cause deadlocks and instability.
4	Poor Lookahead	Choose lookahead analytically. Peformance of a conservative time management scheme is crucially dependent on a good lookahead value. Poor selection of lookahead can explode checkpointing overheads in an optimistic scheme. Several techniques exist to improve lookahead values: stocahastic models, compile and run time analysis, and semantic information.
5	Load Imbalance	Evenly distribute the computational load across all processing units. Static and dynamic scheduling algorithms are available, with dynamic being preferred. It allows for monitoring of computational load and communications between processes and allows for dynamic reallocation of processes.
6	High Message Traffic	Keep message trafic to a minimum. This is closely related to the load imbalance issue in that the decomposition of a model should be aimed at reducing the message traffic among the partitions. In addition, message aggregation can decrease communication overhead by sending a small number of large messages rather than a large number of small messages.
7	Low Event or Computation Granularity	Process LPs in a simulation with the largest number of safe pending events first. This guideline assumes a conservative time management approach and aims at reducing the time lag due to task switching and cache behavior. For example, when a simulation uses threading, process all events for one thread before moving to another.
8	Low Inherent Parallelism	Recognize the amount of parallelism that applies toward a model. Significant performance improvements depend on the parallelism that is inherent in the application. Determine the potential parallelism of a model before wasting effort tuning other simulation-specific parameters.
9	High Checkpointing Overheads	Use an efficient checkpointing implementation for optimistic time simulations. Recording the entire state of a simulation system at every time step can waste precious time and memory. Rather consider interval checkpointing or incremental state saving to reduce the time and computation effort spent on these management overhead costs of an optimistic simulation.

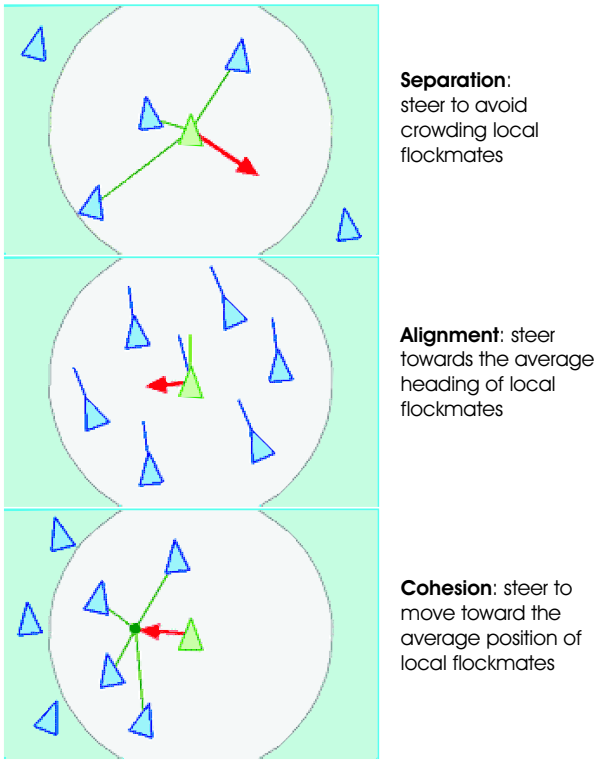


Figure 8 Reynolds' Distributed Behavior Model

1.2.2. From those natural occurrences one can derive a common basic behavior for any kind of swarm. That basic behavior is a small piece of the contribution made by Craig Reynolds in 1987 [85]. He introduced the concept of computer animation of swarming behavior. His flocking model is based on three simple steering behaviors as described in Figure 8–reproduced from [84]. The weighting and implementation of those steering behaviors distinguishes differing swarm models. In his research the objects that are flocking are called 'boids', so often his model is also called the boid model. The figure shows a dark grey circle which represents what he calls 'local' in the description next to each behavior; it is the same concept of a swarm particle's neighborhood.

3.4.1.1 Swarm Models. Not all swarming models are developed the same way, but they all must include the concepts of the three steering behaviors. Below are the alternatives considered in this research. Each alternative has a heading indicating the area of research to which the swarm model is applied.

Particle Swarm Simulation. Discussed in [103] is a swarm model based on particle simulation codes (PSC). The original use of particle simulation codes model the interactions of plasma particles and electromagnetic fields. This large scale simulation follows the orbits of thousands to millions of interacting plasma electrons and ions along the time continuum. These benchmarked, proven stable, efficient, feature rich, and accurate PSC models are reasons the authors in [103] justify PSC as a foundation for their swarm model. Standard force equations and equations of motion model the interaction between swarming individuals. The remaining model elements include center-of-mass, swarming force, friction, dissipation, aerodynamics, gravitation, thrust, obstacles, boundaries, terrain, and weather; these are handled by vector-addition to influence swarm individuals.

Physical Robots. From [64], a behavior based robot experiment explores the pseudo-swarming ideas in the physical world. The underlying model is agent based and assumes the following basic behaviors: avoidance, following, aggregation, dispersion, homing, and wandering. Each agent additionally has a predefined set of goals. A collection of robot agents interacts to produce a group behavior. The agent goals influence the basic behavior algorithms through addition and combination rules. A basic flocking behavior is achieved through summation of avoidance, aggregation, and wandering–strikingly similar to the three steering behaviors described above. More complex behavior is achieved through temporary combination operators and results in "foraging" behavior. This is an excellent model for the researcher interested in empirical observation of swarming behavior.

Communications Model for Swarm Based Sensors. Kadrovach designed a swarm algorithm (see [54]) based directly on Reynolds three steering behaviors, with scalability improvements. The elements of cohesion, avoidance, and attraction are enhanced with the concept of a visibility model. He observes that flocking birds only use a certain visual periphery of influence to adjust their position in the swarm (also noted by Reynolds in [84]). Building on that observation the behavior of one member of the swarm is influenced by only those members within a fixed angle

of perception. Scalability is achieved because the algorithm complexity does not increase as the size of the swarm increases.

Control Systems for Swarming UAVs. Two developments exist that explore the control of swarming UAVs. Both of these approaches include a strategy not yet mentioned in this research: evolutionary algorithms. Optimization is a typical application of evolutionary strategies [111], and that is also the reason that both efforts discussed below include it.

Distributed Control of a Swarm of UAVs. Lotspeich in [62] integrates several additions into his swarm algorithm that is derived from [54]. His goal includes interactions not only within the swarm but also with the environment. Contained in the boundaries are threats such as enemy weapon systems that need avoided as well as locations of interest which are considered goals. The main algorithm for the swarm includes the fundamental concepts of cohesion, separation, and avoidance so that the result is a swarm of UAVs as defined by Reynolds. The additional features include not only avoidance of other UAVs but avoidance of enemy radar sites which results in a global swarming behavior that moves around the obstacle (Reynolds' initial research also included this ability). The attraction forces are used to determine the path of the swarm in the environment. It is assumed that a predetermined map is available for every territory. One drawback to this algorithm is its reliance on the "weighting" of certain variables that is only accessible through an evolutionary algorithm. Thus to achieve greater swarm coherence or avoidance, the parameter set must be "evolved" through hundreds of iterations of the evolutionary algorithm. If the predefined parameter sets are acceptable to the user, this is a good model to follow; however, if a change is needed the price is a lot of undesirable overhead.

Evolutionary Swarm Intelligence for the Control of UAVs. An agent-based model is presented by the authors of [29]. The "evolutionary" portion of the work is focused on fine-tuning key parameters of their model. UAVs move based on sensor inputs of which there are five that are able to sense targets on the ground, other UAVs within a circular region, pheromones, GPS positioning information, and terrain boundaries. The heart of the algorithm fo-

cuses on the "pheromone" strategy which is derived from another phenomena in nature from ants. Ants communicate indirectly by leaving a pheromone trail of where they have walked. Similarly, this algorithm uses the three steering behaviors based on a combination of the neighborhood of UAVs and the pheromone trails encountered. Inferred from the algorithm is a goal to direct a search strategy of the given environment. This swarm does not mimic what is seen in nature, rather it is instrumented to maximize the coverage of the desired search area. Implied in that statement is a lack of expected emergent behavior. Rather than seeing behaviors typical of flocks, what is seen is a methodical search of a given area. Each UAV starts at a single location and branches out, yes avoiding other UAVs, but they have no attraction or cohesion so that a simulation of 10 UAVs results in 10 different directions.

Synchronized Multi-point Attack UAVs. The authors of [63] present a swarm model that is state-machine based. At any one time, the UAV's behavior is dictated based on its current state. Those states as presented in [63] include: avoid, attack, orbit_station, orbit_target, and search. The algorithm model includes an exception–if the current state of the UAV does not provide adequate information for the next action, then a higher layer takes control of the UAV and affects its behavior. Thus, an important part of this model is its layered control system. Under normal circumstances, sensor data provide feedback to the state machine in order to create the behavior of the UAV. Long range communication is assumed. The behavior states produce three produce possible maneuvers: avoid, attract, and orbit. These maneuvers are not consistent with three basic steering behaviors, but rather depend on external swarm objects. The avoidance maneuver is detection of obstacles near itself–which can include other UAVs; however, the attraction maneuver is only toward a target-not other UAVs. Similarly, the orbiting maneuver is not related at all to the steering behavior of swarms as realized in nature, but is a condition invented for the purposes of the synchronized attack model. The attraction and cohesion elements of this model are only partially included and focus on targets rather than other UAVs.

3.4.2 Supporting Models. Several models are needed to produce an end-to-end swarming reconnaissance simulation. This section lists those models in order of priority as defined for this research effort.

3.4.2.1 Communications. Swarming behaviors implicitly require some form of communications. Communications can be used by swarms in passing positional information, data processing, or data fusing. This model places certain restrictions on the bandwidth, range, and response time of the communications that the swarm can handle. Due to the untethered physical model of the swarm members a wireless system is necessary. Because a swarm exists as multiple entities in a local area a communication network can be utilized. Because of the positions the swarm members at any one time are in a state of flux the corresponding network topology also must change. This type of network behavior is exactly like a wireless ad-hoc network. Thus a communications model that works for wireless ad-hoc networks is desirable.

3.4.2.2 Vehicle. A vehicle must interact with the natural forces in a physical world. For this simulation the modeled vehicle is an Unmanned Aerial Vehicle. This model is limited by the laws of physics. The ideal vehicle model includes abilities to interact with the environment in three dimensions– as the physical world exists, thus the flight dynamics are represented in pitch, yaw, and roll angles. Interesting properties for this model include position, velocity, acceleration, structural damage, and fuel count. An algorithm that calculates vehicle movement necessitates a distributed model thus blending well with the swarming algorithms as discussed above.

3.4.2.3 Sensors. Reconnaissance cannot be performed without some way to monitor the surrounding environment. Sensors are connected to the vehicle model and interact with the communications and environment models. Several types of sensors exist for military applications: infrared, radio-frequency, laser, and navigation. Ideally radio-frequency (RF) sensors are modeled for use with this research. The fidelity, sensitivity, range, and other associated characteristics of the RF sensor (commonly called a radar) determine the modeled performance of the sensors in this research.

Processing the sensor information is an extension of the sensor model because of the data collected at the sensing mechanism. The sensor data processing model for a radar model must be able to identify, detect, and track targets. This implies several modes of operation for the RF sensor and associated data processing.

3.4.2.4 Search. An area of interest bounded by a fixed coordinate system results in a finite amount of space. The task of reconnaissance is to perform a search operation within that finite 3D space. How does the swarm move around the area to efficiently and effectively provide confidence of the situational awareness of the targets within that space? The answer depends on the goals of the reconnaissance mission. Refer to Table 4 for a sample of specific goals pertaining to reconnaissance scenarios. For a defined search objective, a search model defines the design of how to explore a search space. For example, with a total coverage objective, the search can be accomplished in parallel with each individual UAV exploring a partition of the search space until all area has been explored. Search optimization is itself an entire area of research [111], thus the model used in this research must only perform a search operation, not necessarily the most efficient or effective.

3.4.2.5 Environment. The features of the 3D space include terrain, boundaries, obstacles, atmospheric conditions, interference, noise, frame of reference, and the laws of nature. That environment model can be as simple as a coordinate system to something as complex as the world in which we live. All other models interact with the environment model because it is the container in which all others exist.

3.5 Parallel Computing Design

Simulating all of the above high fidelity models is done efficiently with parallel computing techniques. Parallel computing takes advantage of HPC resources through the use of distributing processor intensive pieces of a program across several distributed processors. Parallel computing inserts parallelism into a program at many different levels. At the entry level to parallel computing multiple programs can be run independently on separate machines. A more advanced level enables

the same program to be run independently on different machines but with differing data sets. To accomplish higher objectives the parallelism is built into the program algorithms themselves. This level of parallel design requires synchronized communication and algorithms but yields the most benefits.

Parallel algorithm design and data decomposition strategies can be applied to increase efficiency and provide effectiveness that could not otherwise be achieved. For example, using a single UAV platform such as the Predator to characterize an enemy's battlefront presentation would require multiple vantage points from which various resources could be detected through the use of on-board sensors. Using a parallel strategy (i.e. micro-UAV swarm) to accomplish this same goal would increase probability of success through sheer numbers as well as distributing the sensing task among the participating nodes. Decomposing a battlefield into either a rectangular grid coordinate system or an application based organization allows for the swarms to divide and conquer the region of interest in an efficient manner.

Another distributed processing technique is dynamic load balancing. For parallel processing, when a node runs out of work it queries neighboring processors or a master node for more work thus accomplishing more work in less time while maximizing use of resources. Similarly, when a swarm of sensors detects a region of high density it can call on under-utilized nodes to keep the system from being overwhelmed. This section discusses these parallel concepts as applied to swarming reconnaissance.

3.5.1 Data Structure Decomposition. Dividing a computational task into smaller pieces that can be scheduled to run concurrently on multiple processors is the key when designing parallel algorithms. This division can be done by decomposing the data structures on which the algorithm operates and then scheduling multiple tasks of a computation simultaneously. Depending on the application several decomposition techniques can be used. In Chapter 3 of [43], recursive decomposition, data-decomposition, exploratory decomposition, and speculative decomposition techniques are discussed. In [36], several domain decomposition strategies are given for solving partial differential equations but can be applied to other applications. They include structured grid

decomposition, unstructured grid decomposition, recursive coordinate bisection, recursive graph bisection, and recursive spectral bisection. In Chapter 11 of [57] similar domain decompositions are presented as well as striped partitioning and scattered decomposition techniques.

How does one determine the appropriate technique for swarming reconnaissance? If the search space can be represented as a matrix the techniques mentioned above are appropriate. However, if the purpose of the decomposition is applied to the swarm communications problem to optimize the throughput and minimize the latency, then one can consider this an optimization problem in which exploratory decomposition is appropriate because the underlying computations correspond to finding a solution in the search space. If on the other hand one considers the finer level detail of the simulation of the swarm network then the problem turns into a discrete event simulation of the packets traveling across a network topology–at which point speculative decomposition is appropriate because the program may take one of many possible computationally significant branches depending on the output of other computations that are predecessors. A third possibility addresses swarming reconnaissance of a prespecified coverage area which can be measured by grid system thus the domain decomposition techniques, in particular, recursive spectral bisection is the best choice because it yields connected partitions that are well balanced [57]. The answer depends on the selected models to be implemented and associated model fidelity. Ideally all of the aforementioned techniques are applied to the swarm reconnaissance simulation.

3.5.2 Task Structure Decomposition and Scheduling (Load Balancing). Once the domain has been decomposed the parallel machines can be scheduled to start solving their portion of the problem. The idea is to balance the load evenly across all processing elements. Scheduling these decomposed tasks can itself be a whole new problem because of the possible inter-dependencies within the problem domain. This results then in a task-dependency graph also called a directed acyclic graph (DAG). In Chapter 23 of [19] a description of the scheduling problem as being an NP-complete problem is presented as well as algorithms for the static scheduling problem. There are two (often conflicting) objectives when mapping the tasks onto the processors:

1. Reducing the amount of time processes spend interacting with each other

2. Reducing the total amount of time some processes are idle while the others are engaged in performing their tasks

Several techniques are introduced to address this problem of task scheduling and fall under two general headings: static mapping and dynamic mapping. Static mapping is often used in conjunction with a decomposition based on data partitioning and include array distribution schemes, task or graph partitioning, and hierarchical mappings as discussed in Chapter 3 of [43]. Also mentioned are schemes for situations in which the task-dependency graph is dynamic and these dynamic mappings are either centralized or distributed. Asynchronous round robin, global round robin, and random polling are some special dynamic load balancing schemes that work well in splitting the work up among idle processors as discussed in Chapter 11 of [43]. An example task schedule for a reconnaissance swarm application is decomposing the tasks into one of three types: position updates, search strategy and bookkeeping, and sensor data collection and processing.

Shah in Chapter 14 of [19] also discusses the dynamic load balancing techniques that were introduced in [43], but also mentions additional simple load balancing methods: weighting, hashing, least connections, minimum misses, and fastest response. Advanced balancing methods use a combination of the simpler techniques to offer a more useful or practical implementation. Optimizing the balancing method toward one or more of these vectors is accomplished through advanced balancing methods: network traffic optimization, fair load distribution, network route optimization, response latency minimization, administrative or network management optimization, and application-specific performance. These advanced techniques use node traffic-based, network traffic-based, and node load-based balancing as well as load-balancing DNS, topology-based redirection, policy-based redirection, and application-specific redirection [19]. These kinds of techniques can be applied to the simulation framework on which the swarm is running.

3.5.3 Communication. The means of communication is an important aspect when considering parallel computations on HPC systems. How two processors communicate depends on the interconnections that connects the two processors as well as the software mechanism that is used to send messages across the interconnect. Common interconnection networks are arranged as

hypercubes, completely-connected, star, binary trees, and 2-D meshes as described in Chapter 2 of [43]. A thorough presentation of the communication costs with these various network topologies reveals that there are several cost-performance trade offs when considering one interconnection network over another.

The selection of the software that utilizes the interconnection network is also important. There are endless packages available for purchase as well as open source varieties–MPI, Open-MPI, JAVARMI, RTIKIT, CORBA, GM-MPI, and PVM are a few. All of these packages provide constructs that allow communication from the socket layer to a higher level communication interface such as JavaRMI. Depending on the application one package might be preferred over another. For instance, MPI has associated MPE libraries that allow the user to write visualization code right along side of the parallelized source code to enable a communications visualization program. Other differences include taking advantage of heterogeneous architectures–i.e. JAVARMI and CORBA–for example relaxing the requirement that all processors have the same operating system. Selection of this software is determined in the next chapter.

3.5.4 Speedup. The goal of parallelizing an application is to improve its wall clock time and memory capacity. But how much better does the parallel program do when compared to the best known serial implementation? The speedup equation captures the benefit of solving a problem in parallel vs. serial. The formal definition of speedup, S, is the ratio of the serial runtime of the best sequential algorithm to the time taken by the parallel algorithm to the same problem on p processing elements. Equation 4 is the equation for speedup,

$$Speedup = \frac{T_s}{T_p} \tag{4}$$

where T_s is the fastest known time to solve the problem in a serial manner and T_p is the time it takes to solve the problem in a parallel manner. The reason why the best serial implementation is chosen is because there are some serial algorithms that cannot be parallelized, so to not include them would give skewed results toward parallelization [43].

3.6 Visualization

Visualizing the swarm is the final aspect of design for this chapter. Visualization plays a subtle but important role in swarming behavior. After any model is developed, its merit is established based on well-known benchmarking datasets or tool sets. Without proper validation or corroboration from fellow researchers the model adds nothing to its research community. Therefore, the question of validating a swarming behavior becomes an important one. The correct answer is not a difficult one, but rather obvious. Because swarming behavior is a natural occurrence anyone who has experience with the natural world can validate the model assuming that it provides visual output. This is where the rub is. If a model does not visually represent the simulated interactions between objects as they occur in the algorithm, then it is incorrect and should not be used. For example, in Kadrovach's swarming algorithm several interactions (position updates) occur between objects that are not displayed in the visualization. This means that all movements that are not visible occur in a dimension outside of our known universe. In short, it can be stated that a swarm algorithm cannot exist independent of a corresponding visualization system or a well-defined interface to a visualization.

This section annotates the ideals for a desired visualization based on application specific parameters; that is, features that an application should provide or let the user have control over. There are several elements of this research that can be visualized such as network communications, topology, swarm behavior, associated metrics, parallel processing communications, and an integrated target, sensor, swarm picture. Because this paper focuses on a reconnaissance mission the latter is the primary subject of visualization. Specific visualization requirements are categorized under various properties of a visualization system.

3.6.1 Data requirements.

1. Generation: Although highly desired, the ability to generate high-fidelity data is not a requirement of the visualization package. That is the task of the simulator.

2. Import: Data output from the swarm simulation system must be readable by the visualization package.

3. Export: Once a visualization has been developed for a target audience, it is necessary that the software package be able to produce high-quality outputs of the visualization. This includes lossless compressed formats for visual images (i.e. tiff , bmp, or png image formats) as well as standard streaming video formats for animation sequences (i.e. mpeg, AVI, or mov movie formats).

4. Format: Imported data must not be constrained to fit a specific data structure. Rather, an efficient interface should exist that allows the user to specify column and field definitions.

3.6.2 Color utilization. For the human eye, light is a reflection of not only the item of focus but also of the surrounding canvas. Light is relative [108]. Because an object's appearance is different depending on the relative contributions of neighboring object's appearance, it is important that UAV objects be presented with this in mind. Perception has many factors that comprise the way our eyes see light–each requirement is outline below.

1. Brightness is the perceived amount of light coming from a source, thus for this case study, the brightness level should be comfortable–not overwhelming the user. This is more a function of the media on which the visualization is presented. Thus the chosen brightness patterns must be adjusted for presentations made on poster board, printed paper, and video projections.

2. Luminance is the measured amount of light. This channel of the human visual system is fundamental to perception. Ware suggests compliance with the International Standards Organization (ISO) specification 9241, part 3 [108]. It recommends a minimum 3:1 luminance ratio of text and background with 10:1 being preferred. Implied in this recommendation is that gray-scale color is not the best for encoding data–because of the smaller luminance ratios between grays. Thus utilizing colors with a wide variety of luminance increases one's perception. This ISO standard shall be the requirement not only for text and background difference, but also for high fidelity areas of interest.

3. Contrast effects can produce illusions that perceptually make some colors and shades look like different shades due to the surrounding color content and brightness. Errors can result from perceptually recognizing the wrong relationships between neighboring colors, thus contrasts should be done with ranges that avoid these errant effects. For gray scale images the acceptable contrast levels are grey, white, and black–only three. For color, the gamut is much larger–twelve are recommended for use in coding because they are reasonably far apart in color space.

4. Saturation is defined by scientists as a term to denote how pure a color seems to the viewer[108]. In an effort to have as much control as possible over the visualizations the software package should be able to directly apply a level of saturation to various elements.

5. Chromacity defines the hue and vividness of a color while ignoring the amount of light. Fine details shall be visible through the use of appropriate components of luminance and chromacity. These details should not be displayed with purely red-green and yellow-blue chromatic channels, rather there shall be considerable luminance contrast (black-white) in addition to color contrast.

3.6.3 Glyphs. One way to represent multi-variate discrete data in many dimensions–glyphs. This is done by using a single graphical object and mapping the multi-attributes of interest to the various characteristics of the object–size, color, position, shape. This information representation must be available to display targets with relevant contribution data from jamming, ground clutter, and noise.

Integral dimensions present encoded information in an integral format, meaning a holistic approach[108]. Decision making based on integral dimensions are made with not just one variable in mind, but all the contributions integrated together. The various contributions of the return radar signal for each target (true and false) show the use of integral dimensions to indicate combined contributions for each target. Separable dimensions are also employed by using different shapes. Thus both integral (combined color) dimensions and separable (different shapes) dimensions should be used with the targeting visualizations, depending on the relationships that are being portrayed.

3.6.4 Notes on Encoding Data. Even though several methods are specified for encoding information in a visualization that the number of encoded elements (whether glyphs, iso-surfaces or x-y plots) should not overwhelm the viewer and thus the total encodings should not exceed seven for a given image. This prevents strain on the analyst who is performing the visualization task.

In addition, flexibility is the key to producing an accurate representation of the data. With the use of color, glyphs, surfaces or combinations thereof for mapping scientific data, one can easily produce a visualization that tells a different story than reality[87]. Therefore, the software package should have sufficient flexibility to allow an accurate representation of the data and its structure by controlling the behavior of these visual cues that encode information into the visualization.

3.6.5 Computational Steering. Visualizations involve a greater level of interaction with the simulated (input) data [46]. The idea is that the visualization is tied inherently to the simulation in progress, where for our study, the simulation is that of swarming reconnaissance. A real-time computational steering capability is important for the design-test-design process. During implementation, a working visualization with real-time computational steering of the swarm behavior gives immediate feedback on design changes. The advantage of being able to computationally steer the perspective involves seeing the swarm and environment from various perspectives at any time during the visualization. This concept also applies to visualizing the sensor data within the swarm as well as the data processing of the sensed information.

3.6.6 Interpolation. Interpolation is the fundamental process that is used to create an empirical model of the phenomenon that is being visualized. Simplistically, choosing a method of interpolation is choosing the manner in which the unknown data between the known data points is calculated. It is understood that various interpolation methods are used to construct textures and other visual enhancements, however, the impact of these methods directly affects the perception of behavior within a swarm. This aspect of the visualization package should not be underestimated but rather be given careful consideration as it critically impacts the validation of swarm modeling. An excellent resource to consult for interpolation methods is [17].

3.6.7 Focused Interest. The ability to identify regions of focused interest based on parameter values is a valuable tool for analyzing complex data. For example, probing particular swarm members for current position, sensor identifications, target list, and associated vehicle properties is an acceptable implementation.

3.6.8 Animation. An animated sequence visualization capability clarifies to the viewer the behavior patterns of the UAV objects as they interact within the swarm. A package without the ability to animate is lacking a feature critical to the success of designing a swarming reconnaissance application.

3.7 Summary

This chapter discussed important high level design aspects of a parallel swarm reconnaissance application. Development of the reconnaissance scenarios are influenced by enemy poise and threats, number of targets, type of terrain, mobility, swarm size, world dimensions, and mode of operation. How a swarm performs in each scenario is captured in specific MOPs and MOEs. Next a comparison of optimistic and conservative time management approaches for parallel discrete event simulations reveal that one strategy is not necessarily preferred over another. The algorithm section defines the basic swarming model preferred for this research and outlines current research in existing swarm design. Augmenting the swarm model are the design requirements of supporting models that are needed to complete swarming reconnaissance design. The next division highlighted design strategies for applying the model to a parallel computing environment. Finally the importance of visualization as it relates to swarming is argued and then followed by swarming visualization design concerns. The next chapter documents the task of implementing a swarming reconnaissance model.

4. Low Level Design and Implementation

4.1 Introduction

The previous chapter annotated design principles, guidelines, and model ideals that this chapter uses during the low level implementation of the swarming reconnaissance model. This chapter covers implementation related design considerations that are part of the software engineering effort for this work.

4.2 Simulator Selection

Commonly taught among software engineering institutions are techniques that increase the software quality by improving code maintainability, reliability, re-usability, testability, usability, traceability, learn-ability, and portability [16]. Code reuse is one way to gain the most benefit across the range of software quality measures. Simulating a swarming reconnaissance mission as shown in the previous chapter has many complex associated models. To write all of those models from the ground up is a task for the individual without time constraints. Therefore, motivated by software quality and reasonable expectations a selection process occurred which analyzed existing software models and simulation frameworks pertinent to this research effort. That study is provided in Appendix B.

The study of a simulator for a swarming reconnaissance mission included such aspects as communications, vehicle movements, sensor characteristics and fusion, swarming behavior, target descriptions, and environment models. Three categories of simulators were evaluated against the desired set of prioritized characteristics: swarm simulators, network simulators, and simulation frameworks. It was found that none of the simulators encapsulated all of the desired simulation traits. This led to the decision to use a layered selection approach where the SPEEDES parallel environment was chosen as the lowest layer and the initial two component models to be selected are Kadrovach's swarm behavior model and the network simulator, *ns2*. This leaves models to be designed for integration into the SPEEDES framework as work continues after this research. While finding an all-in-one swarm simulator for the characteristics described in Appendix B did not happen, this effort provides the beginnings of a future comprehensive swarming reconnaissance simulator.

To recap, the following existing simulation models are chosen:

- SPEEDES: Chosen as the parallel simulation backbone
- kswarm: Kadrovach's swarm algorithm as implemented in C++ as the swarm algorithm model for this research
- *ns2*: Berkeley's network simulation model as the communications model

Integrating these simulators into a working swarming reconnaissance model is necessary for validating the measurable objectives as stated in the first chapter of this document. The sections that follow describe the process of moving from a conceptual design to an implemented program with the selected simulators.

4.3 SPEEDES PDES Framework

An overview of the Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES) is presented in this section along with the specific design considerations given toward the swarm implementation.

4.3.1 SPEEDES Overview. This C++ open source package is a general purpose parallel discrete event simulation framework that is currently being maintained by Metron Incorporated. It has an extensive set of built-in algorithms, data types, and utilities that lighten the programming load and increase parallel performance. A formal overview can be found in [50] and a detailed reference is available from [69].

The main aspects of SPEEDES apply to the swarm simulation are the basic object structure, the object interaction through proxies, data distribution, simulation algorithm, and external interfaces. The object structure refers to how simulation objects are created. All must inherit from a built-in simulation object that is SPEEDES-aware. The defining of new classes/objects is accomplished through extensive use of macros or "#define" functions and statements. A SPEEDES simulation is simply an ordering of events that are created and scheduled by user-defined objects. Primitive object interaction is event-based and much liberty is provided in specifying how those events are defined. For instance one object can schedule a certain event based on a specific simulation time or a process can be defined that continually creates and schedules events based on the

process algorithm. Data proxies/ distribution management is the vital portion of the SPEEDES implementation that allows objects to have 'relationships' with other objects in that they share certain data fields with each other during the simulation. These relationships are defined by a publishing/subscribing/discovery scheme between simulation objects and classes. An even more powerful use of the relationships comes from the data distribution algorithm. It allows objects to subscribe at the attribute level and then even in specific ranges of those subscribed attributes, so that only the needed data is seen.

The simulation algorithm can be specified at runtime as optimal, conservative, or somewhere in between. The optimal simulation algorithm uses a Breathing Time Warp (BTW) algorithm that strives to balance cascading anti-message explosions (risky event processing) with too many synchronizations (no-risk event processing) [69]. The BTW is the cornerstone of the SPEEDES framework and thus is the driving reason for much of the design. For instance, SPEEDES introduces a supporting concept of "rollback" types that are an inherent part of the BTW, thus all state variables in a simulation object must be "rollbackable." Finally, external interfaces allow SPEEDES to talk with other applications. The interface is done through a state manager which allows the outside program to send and receive data and start and stop events in the simulation queue.

Below are given highlights of the SPEEDES framework as they have been applied to this research. While much of the framework is used, there is still an amazing amount of simulation functionality built into this framework that has not been used with this research that focuses on efficiently implementing a discrete event simulation including but not limited to land, sea, and air vehicles. Read the manual for a detailed introduction [69].

4.3.1.1 Events. A simple way to create causal relationships between simulation objects is through events. Object A can schedule an event on itself or on Object B at a time in the future of the simulation. This is the most primitive and inefficient way of generating relationships among simulation objects, however, it is simplistic and still effective. The event model is used in the "working" SPEEDES implementation of Kadrovach's swarm model.

4.3.1.2 Proxies. One of the fundamental concepts in the SPEEDES framework is the ability to share data between different simulation objects. While common discrete events

provide similar functionality, a proxy is a more efficient method for situations when this data is shared repeatedly. SPEEDES calls this ability of one simulation object to see another's state data "Object Proxies." The public part of the object's state data is automatically mirrored to all proxy holders whenever its value changes. Applying this concept to a parallel swarming application, each independent UAV simulation object is calculating updates based on public state data of neighboring UAV simulation objects. Proxies are used to announce when any neighbor has changed its public state attributes, i.e. position, direction, velocity.

4.3.1.3 External Interfaces. A bonus feature of the SPEEDES framework is the ability to communicate to and from any simulation object outside the simulation itself. This is done through what SPEEDES calls an "External Interface." External interfaces can be seen as simple proxies that an outside program is holding. In that manner, data can be easily passed to an analysis or visualization program that can process the current object state data. The main difference between an External Interface and a regular Object proxy is the process that holds the proxy. If that process is internal (i.e. another simulation object) then all data changes are propagated, however, if it is external (i.e. a program that has its own main( ) function) then it only receives committed data changes, thus it lags the current simulation time, as expected. This research uses an External Interface to pass real-time position updates to a visualization system so that the user can see the swarm in motion as it has just occurred. While this is not a real-time interface, it can interact with the simulation causing events to be rolled back and other events to occur that would not normally happen.

4.3.1.4 Data Distribution Management . Proxies are also a main supporting backbone for the Data Distribution Management (DDM) function of the SPEEDES framework. DDM is an advanced feature of the framework that allows filtering on the sending side of the proxy rather than the receiving. When a simulation object holds a normal Object Proxy every single time a change occurs in the state data it is propagated. There are times when this is undesirable, for example, when modeling a UAV's communication mechanism. In that case, each UAV in the swarm could hold a proxy for the whole class of UAVs, so that the state data of every UAV simulation object is passed to every other UAV. While this would be an ideal example, the real world requires limitations on the range of wireless communications, thus DDM applies. Each UAV has DDM

proxies for every other UAV, but this time only those UAVs within my communication range are passed to my simulation object so that I do not have to filter out the "out of range" UAVs state data. In a situation where hundreds or thousands of UAVs are involved DDM greatly reduces the required communication overhead.

4.3.1.5 Breathing Time Warp Algorithm/ Rolling Back. The Breathing Time Warp Algorithm is discussed in detail in [97][98][96]. It depends on the concept of state data that is resumable or "rollbackable" as annotated in the SPEEDES manual. An application of a rollback is when one event is processed ahead of another, for example a UAV in the swarm moves to its position at t=3 while at time t=2 there was a UAV that had already moved based on the old position of the first vehicle. In that situation at the time the first UAV moved the SPEEDES framework recorded the associated event that caused the change in its state, so that when the second UAV move causes the first one to be nullified the framework can reprocess the tagged event propagating the correct information for the remaining events. To make a state variable rollbackable one must use a predefined class in the form of RB_int or RB_double.

4.3.1.6 GridManager (DDM specific). This subject is mentioned briefly in the user's guide to SPEEDES, but it is of great importance especially when considering the communication overhead of running this simulation on a Beowulf cluster rather than a shared memory architecture HPC. Insight into this importance was gained from a visit to AFRL/IFTC where there is a large SPEEDES user base. The GridManager keeps track and processes hierarchical grids. An example distribution is found in section 11.6.2 of the SPEEDES manual. These are simulation objects used to optimize the DDM within SPEEDES so that it scales in both memory and number of objects. According to the manual (11.7.1), creating more hierarchical grid objects results in fewer rollbacks, but this is done at the expense of a larger memory footprint.

4.3.2 Algorithmic Models. Porting existing model implementations to a parallel framework involves many design decisions. Often the implementer is faced with choosing one design principle over another when integrating the models. The foundational model of this research is the swarm behavior model. The next model of interest is the network model. Finally, the remaining support models provide functions not available in the swarm or network models.

4.3.2.1 Swarm Behavior Model. How well does the Kadrovach's implemented swarm behavior model meet the requirements as stated in the objectives of this research? The answer is satisfactorily. The disadvantage is twofold: the libraries used are not publicly available source code and the implementation is designed to run on one CPU. The advantages include that the model is a correct representation of the selected swarm model for this research, it is coded in C++, and the documentation is substantial.

Porting to Linux. One major set-back to selecting Kadrovach's implementation is the number of Microsoft Foundation Class (MFC) dependencies. Several underlying data structures in Kadrovach's implementation inherit from many of the objects in the MFC library. Another challenging aspect of this implementation is the target platform was a serial machine–single CPU. Thus, all the code is focused on one process.

Porting this implementation into the SPEEDES framework is done incrementally. The first increment is porting the program to the Linux operating system. The second increment is then to port the program into the SPEEDES framework. This second increment is further subdivided into two more parts: port the serial implementation exactly as it behaves in the serial model, then that model is enhanced with features of parallelism.

The first increment is accomplished by identifying those objects that are Microsoft proprietary and replacing them with an equivalent Linux variant. The ported classes are shown in Table 7. The first column indicates the the reference to the first place where the class is used in the code and the second column shows the number of total uses within the code. The third through the fifth columns provide the name changes and descriptions of the converted classes. Simply changing the class name and including a header file did not solve the compilation errors. One of the most difficult challenges was figuring out the depth of a pointer used by a template class and being able to traverse the template list while providing the correct pointer. A commonality for all replacement classes is the "Q" leading each name. This is a trademark of Qt, which is a C++ toolkit for multi-platform GUI and application development [104]. This was a prize find because all of the Microsoft classes could be converted to Qt equivalents with little effort. These classes are freely available to the Linux community.

Table 7 Converted Classes Used in Kadrovach's Swarm Simulator

variable	#	Microsoft Version	Linux Compatible	Description
CFormation:: m_listMoves	1	CTypedPtrList <CObList, CStep*>	QPtrList<CStep>	Template class that provides a list of Cstep objects
CFormation:: findistEx:: tarray	7	CArray<CParticle, CParticle>	QValueVector <CParticle>	Value-based template class that provides a dynamic array of Cparticles
CFormation:: SetRegionSize	1	CSize	QSize	Class that defines the size of a two dimensional object
CFormation:: GetIndex:: rectRgn	2	CRect	QRect	Class that defines a rectangle in a plane
CFormation:: toPoint()	9	CPoint	QPoint	Class that defines a point in the plane
CFormation:: setDynParams	1	CString	QString	Class that defines a string object.
CFormation:: Serialize	1	CArchive	QDataStream	Class that provides serialization of binary data.

Porting to SPEEDES. The second increment required integrating the Q libraries with the SPEEDES framework which was made easy by the qmake tool (part of the Qt C++ toolkit). In addition, several design trade offs need to be considered when using the SPEEDES framework. What attributes make up the states of the simulation objects? Should the optimistic or conservative simulation algorithm be used? What mechanism is used to generate events? The state attributes need to be of "rollback" types if the BTW algorithm is to be used. The mechanism that generates the events refers to the main simulation loop. How should the main simulation loop be distributed across the parallel architecture is the thrust of the question. Initially the thought is to represent each of the swarm members (i.e. particles) as separate simulation objects. When using a distributed design model, a swarm's neighborhood can easily be defined through using the advanced features of the built in data distributions manager so that only those objects within a specific range (or direction angle) would publish data back to the neighborhood's center object. These questions are addressed in detail.

The main simulation algorithm is shown in Figure 9. This algorithm resides in the `CFormation` object and in particular the `moveupdate()` member function. This function is called for each particle in the swarm of size n. This simulation algorithm has a complexity of $O(n^4)$. The SPEEDES equivalent uses the same algorithm, but the outer loop is no longer required. Instead each iteration of that loop is distributed so that n simulation objects perform $O(n^3)$ operations simultaneously (assuming no rollbacks).

Ideally, when porting this serial implementation to a parallel environment, the speedup would be linear, meaning that with each additional processor the time to execute the main loop would be divided by the total number of processors. This is not the case as shown in later chapters. Key design decisions provide context to and answer the questions posed above. They are listed below.

Representation Each swarm member is represented as its own simulation object. This decomposition lines up with recommended parallel design practice.

State The state of a simulation object corresponds directly to the state of one UAV object. Yes, all UAV variables contain "state" information that needs to be declared by the provided macros in the SPEEDES library as "rollbackable." During development,

```
Pseudo code for CFormation::moveupdate( )
for (all particles: i){

    populate neighbors array for particle i sorted by distance: O(n^3)
    process boundary influences
    for (all particles: j); j ∈neighborhood_i
       is particle j visible by particle i
    for (all particles: j)
       if visible then process particle j's influence
       move particle i to new location
       update distance matrix
}
increment simulation time
```

Figure 9 Kadrovach's Main Simulation Algorithm

the importance of this design decision became evident after hours of debugging line by line in sync with the working serial program. The variables that are rollbackable in the current implementation are the UAVs position data (x,y,z,d) and the local ParticleArray data structure. Use the optimistic time management algorithm-BREATHING TIME WARP!

CFormation Kadrovach used this class along with the main program to implement the loops that iterate through each member of the swarm and update the current particle's position and distance matrix. This class was absorbed into the simulation object called S_UAV. Thus, each UAV logs its own local bookkeeping of the formation movements. While this goes against good design principles, distributing that bookkeeping data structure required a fundamental change to the inner workings of Kadrovach's complex behavior model.

Position Updates A global data structure is used in the serial version of Kadrovach's code eliminating the need to propagate updates between swarm members since each particle has direct access to the latest position information. This is not the case in the SPEEDES environment. Because each object is independent then some form of communication must exist that allows for the propagation of position updates. Original thoughts

included using the distributed data management inside SPEEDES to eliminate redundant updates for particles that are not part of a UAVs neighborhood. After the initially pursuing this goal, it was discovered that the same global bookkeeping data structure prevailed in the neighborhood model as well as the visibility blocking model, so that a major code revision was required. Thus, a lesser desired approach was reluctantly accepted (event-based) in order to obtain a working parallel model. Using the DDM function should be pursued by future researchers as it can greatly increase performance and decrease the runtime of a SPEEDES application. Efforts continued to optimize this communication overhead by providing a second implementation selectable at compile time that uses a process and proxy subscriptions to propagate vehicle movement updates. The resultant behavior of this second implementation differs from Kadrovach's serial version in one fundamental way as discussed below.

Writing_swh The swarm history file (*.swh) is written through an External Module program. It is an independent program with its own main function and Makefile. In order to receive position update information this module must communicate with the SPEEDES application. This is done through the SpeedesServer communication program. This file remains in the exact format that Kadrovach used for his research, so all the associated visualization and analysis tools can understand it.

4.3.2.2 Fundamental Differences. Two main implementation decisions differ from Kadrovach for reasons stated next.

Visualization/Physical Representation. Alluded to in Section 3.6 is the idea of a validated swarm model. As understanding was gained about the implementation of the serial swarm model some questions came to mind about the validity of the model that he used. It is not to question the outcome of his research efforts–for those are amazing in there own right! Rather, it is a question of how one separates the swarm behavior algorithm from the visualization of that algorithm. One assumption Kadrovach did not make is that of physically implementing his model in a real world swarm of micro UAVs. A global data structure would never work, but more importantly the concept of time as implemented appears inaccurate. Common to both a visualization system and a physical implementation is the hard requirement that the laws of time be maintained. When

visualizing time-stepped data it is crucial to maintain synchronization with the second hand on the clock or else the results are skewed. Similarly, when implementing a physical representation of a synchronized system those interacting elements must maintain a common clock. In both cases, the serial swarm implementation lacks consistency. For the visualization system, assuming each time unit is equal steps (seconds for example), then any time-dependent movement must occur in sync with one second. That is not the case for the serial implementation.

For example, assume there are 3 particles in the swarm moving toward the east with some velocity. This logical progression reveals inconsistency:

1. Particle 1 (P1) moves to a new position at $t = 1$. According to the serial algorithm, P1 just traveled some finite distance and the time is now $t = 1 + \Delta = t_a$.
2. P2 uses the updated position information from P1, to calculate its new position, however, the algorithm does not advance time for the P2 update to $t = t_a + \Delta$, but rather maintains that both P1 and P2 move at exactly the same time: $t = 1$ even though P1 moved *before* P2 could calculate an update.
3. Thus, when visualizing this swarming algorithm, all of the Δ updates are not displayed, but hidden from the view of the user.
4. One might argue that these Δ updates are indicative of communication delays or other data position updates, however,
5. if P1 moves first then after arriving at the new destination some Δ time later (say at $t_1 = 1$) his new position is sent to P2 at $t = t_1$ so that P2 in turn moves
6. and communicates his update at $t = t_1 + \Delta(P2)$ therefore P2 has moved at time $t_2 = t_1 + \Delta(P2)$ which proves that $t_2 \neq t_1$ which is what the serial algorithm proposes.
7. Thus the algorithm has an inconsistent time model or assumes that positions can be predicted, which in an emergent behavior system is impossible.

A simple change allows for a consistent time representation, which is how the SPEEDES "process & proxy" implementation models the swarm. All that is needed for correct reporting of position time update pairs is the visibility of each position update as well as the time that it took to perform the movement. Thus t_2 is always $> t_1$ when a second particles receives an update based on

a historical activity. After making this change, there were slightly noticeable differences in the behavior model. The flocking behavior showed little difference, but the swarming behavior is considerably smoother (almost flocking). The quick bursting random direction movements are now smoothed into smaller changes in direction and their is much less perceived acceleration. The above discussion is formed on a basis that the swarm members make updates in a serial fashion with relatively equal intervals. That does not necessarily need to be the case.

Position Update Considerations. How are each members swarm position updated? Can the existing algorithm be improved? The answers to these questions dramatically affect the performance of the SPEEDES swarm application. Below are some alternative ways to implement the position update portion of the algorithm.

Position updates in the context of the following discussion implies not only a coordinate pair, but also a heading (direction.) Kadrovach's original implementation of the swarm model calculates the swarm member position updates in a strictly equal time-stepped fashion, which implies that he used a synchronous (time-based) rather than an asynchronous (event-based) simulation. The algorithm that calculates the position updates loops through the entire swarm at $t = 0$ calculating member updates based on the current positions. Thus for $t = 0$, the i^{th} particle is updated based on neighbor positions as defined during time 0. The complication comes when calculating particle $i+1$'s new position. Since particle i's new position has already been calculated, it is available to be used by all subsequent calculations (this is where the illogical time concept happens as mentioned above.) Each particle calculation while using all position information at $t = 0$, if the new position information for $t = 1$ is available and applicable to the current particles neighborhood, then that new information is used. Thus there is a ripple effect, in that the very first particle will always be using $t = 0$ position information, while all remaining particles might be using both $t = 0$ and $t = 1$ position information–assuming it has previously been calculated and applies to the current particle's neighborhood. This is shown in Figure 10 on the top where the updates cascade one at a time down the continuum in an equal stepped fashion (note the clock is paused while all updates occur until particle n finishes, at which point time progresses in equal steps).

A more realistic approach suggested by Kadrovach, would be scheduling events as a position is updated for those in the neighborhood that can perceive that positional change. This would

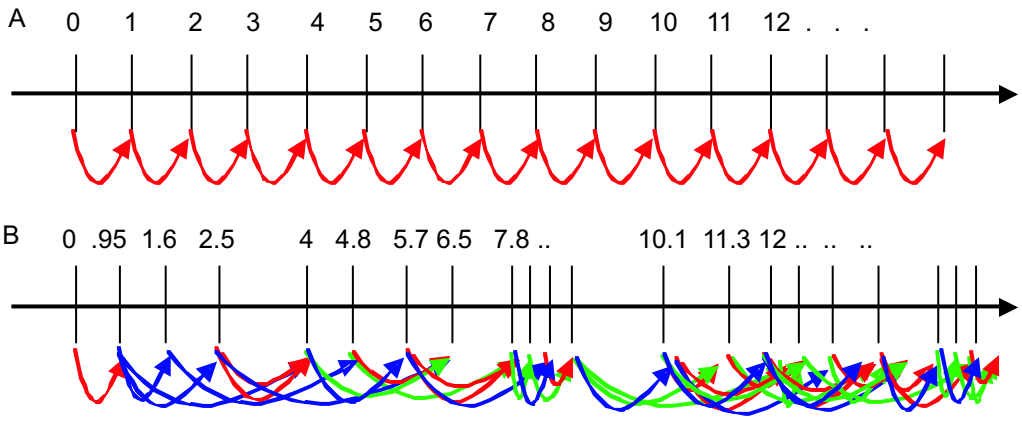


Figure 10 Position Update Illustration

cause a chain reaction for all members of the swarm to cascade updates throughout those in their surrounding community in a non-linear fashion. The position update illustration shows in the serial implementation that an update only cascades a forced effect on the next member in the for loop. However, using the more realistic approach an update can cascade the effects to all neighbors that have the current particle in its visibility window. Thus, in part B of Figure 10, an update does not occur at fixed time intervals (assuming the continuum is time), but is dependent on the communication delay. Also, one update affects all members that perceive the current particle indicating that some updates cause more communication to occur (multiple arrows) than others.

A difficult part of understanding how to implement this shift in position update time frame is understanding how things are started. At first, one might be confused as to how this whole position update occurs as it sounds like circular reasoning; however, what must not be forgotten is that at some discrete point in time the swarm is going to be initialized with one member in the lead and the others following. So the steady-state condition involves updates continuously propagating throughout the swarm, while the initial-state has far fewer updates that are propagated–at least until all swarm members are deployed out of their cargo transport or ground launch. In this way, the initial update occurs from the swarm members who are leading. Because the leaders of the swarm have very few (or possibly no) visible neighbors to influence them their update cycle is more rapid than the others. In short, from the time the UAVs are launched until the swarm reaches a steady-state, the number of swarm members who are communicating updates starts with 1 and

exponentially (depends on the swarm topology) increases until all members are either publishing position update information or calculating it.

Assuming the context is a PDES system, some questions arise as to the validity of the position update method described above. If all swarm members are waiting for position updates before updating their own position how does any one member ever move (deadlock)? How does a swarm member know which time-position to use when updating his own position? To ensure all possibilities have been considered for the swarm behavior the following statements summarize the impact of the position update interval strategy and answer the above questions:

- Coherent Swarm Behavior (Flocking)
  - Two possibilities exist why a particle i has not received position updates:
    - member i has no neighbors, therefore calculate new position now or based on a delta t
    - member i has neighbors but has not received all of their updates yet occurs when:
      - swarm initialization is occurring, therefore wait until all visible neighbor position updates are available (stay on initialization sequence)
      - communication failure is occurring, therefore project using the most recent data for the unknown particle
      - swarm member has been destroyed and is no longer a part of the swarm
- In-coherent Swarm Behavior (Swarming)
  - Updates from neighbors are only used to maintain center of gravity and collision avoidance
  - several starters lead the swarm in varying directions rather than a single member but since these leaders have no visibility they rapidly turn around to maintain a center of gravity proximity
  - the flocking rules still apply, but the initialization time frame is probably shorter because there are multiple sides of the swarm being leaders and thus producing updates

A final alternative for the position update dilemma uses angular information to filter what neighbors are influencing the current particles direction. This approach can be constructed in the SPEEDES environment by receiving position updates through a proxy for every particle in the current particle's neighborhood of interest. Deciding on which neighbor to process first or to process at all can be implemented through a filter that detects angular movement. Thus neighbors that have larger angular movement have greater influence and are considered a higher priority when calculating a position update. A particle then has variable influence with varying priorities resulting in non-linear positional influence–which can be far more efficient than a forced equal interval update pattern as currently implemented.

4.3.2.3 Network Behavior Model. All throughout this document are references to a network model or communications. A swarming model is just as dependent on the communications as it is the attractive, repulsive, and alignment rules. Unknown at the time of simulator selection is the incompatibility between the *ns2* network simulator and the SPEEDES framework–one must be eliminated. The ideal is a combination of the two simulators–extracting and porting from one environment to the other, but this is outside the scope of this effort. Why are they incompatible? The time management algorithms would cause extensive simulation processing delays rendering both simulators useless. Because of this incompatibility, a study examining the possibility of using a parallel network simulator [86] for swarming is accomplished. The *ns2* network simulator is analyzed for parallelization applications. A design of three experiments are carried out to measure the throughput, latency, speedup, and scalability of swarming applications, but the parallelized version has strict topology requirements which prevent its use with a swarming model. Thus, it is concluded that using *ns2* as a communications model to support swarming behavior is not a feasible option. For more detail about this study see [28].

Directed Diffusion Implementation Considerations. One potential issue with using directed diffusion with a swarm-based formation of sensors is the dynamic nature of a swarm. If there are only 2 kinds of movement (see Section 2.3.3) that a swarm takes on, the impact of the issue is lessened but still remains. The problem arises when a swarm is dynamically moving across a target zone and an interst is propagated throughout the formation. When one interest message (see Section 2.2.3.1) is sent with a timeout of value `t` then it is assumed that the topology of

the network does not change such that the requestor of the interest message is no longer within communication range of the sender. Thus the dynamic nature of the swarm is working against the efficiency of the data communications protocol. For example, consider a topology that includes members A through E who are all within communication range of each other. Suppose member B sends a receives an interest from the ground commander and then sends requests to A and C for the specific data with a time limit, `t`. Before that limit is reached the swarm topology dynamically moves such that A and C are no longer in range of B, but at the same time A and C both identify data that matches the specific interest item. Both A and C broadcast their response to the interest but are both outside the communication range of B so when another member receives that data it is ignored because the receiving member never requested it. So the expiration time limit of an interest message hinges on the duration of a network topology. This is one reason motivating a deeper look into the applicability of the Directed Diffusion protocol for use with swarms.

4.3.3 Visualization. Two visualization systems provide visual interfaces to the parallel swarm simulation: Matlab and Skyview. Matlab reads the binary movement history file and shows an animation. This visualization is easily customized thus used for quick data analysis. Much of the code for Matlab came with the original serial implementation so credit is given to Kadrovach for that visualization system. A second visualization integrates the swarm into an existing visualization system available as part of IDAL, called Skyview. This software requires specialized hardware and produces a 3D interactive visualization of the swarm. A benefit of the Skyview package is that it is a near real-time implementation. The displayed swarm is being updated with the advancement of each simulation time step so that the binary history file is not needed. Currently, only the Linux port of the swarm simulator works with Skyview. Skyview is developed and maintained by AFRL/SNZW and provides visualization for any simulation that reports object information according to the IEEE Distributed Interactive Simulation standard [92].

4.4 Summary

This chapter opens with the selection of a simulation system that includes three solutions: SPEEDES, kswarm, and *ns2*. The remainder of the chapter discusses each of those solutions with respect to the swarming reconnaissance model. The important functionality of the parallel

framework (SPEEDES) is summarized as it is used in this research. Next, porting implementation concerns are discussed for the purpose of communicating the challenges encountered during the process. After that, two fundamental differences that were discovered through the development process are discussed. The chapter concludes with a discussion of the implementation obstacles with the network behavior model. The next chapter presents a design of experiments for this research.

5. Design of Experiments

5.1 Introduction

Presented in Chapter 1 are sub-objectives that indicate success for the two objectives outlined for this research. Validating those sub-objectives motivates the design for the experiments written in this chapter. Several basic building block experiments lead up to a final topology/model/algorithm configuration for the reconnaissance mission scenarios. First the parallel discrete event simulation is experimentally configured. Second the parallel swarming algorithm experiments are designed. Finally, swarming reconnaissance experiments are designed. Each experiment associates with one of the sub-objectives stated in Chapter 1.

5.2 Parallel Discrete Event Simulation Experiments

A series of SPEEDES experiments enables one to configure the parallel system for the most efficiency and allows one to validate proper SPEEDES functionality. Details of each of these experiments is presented after a brief introduction to the experiment ideas.

To find an efficient configuration for the SPEEDES framework one can characterize the communications impact that results from the internal communication libraries and algorithms of the parallel discrete event simulation framework. Why only characterize the communications impact? Because when changing from a serial to parallel computing platform, the improved performance is primarily based on the speed of the communications between processes [43]. Given a fixed network topology (cross-bar), fixed data handling and routing (cut-through), and a fixed protocol (TCP) the variable becomes the process communication algorithms. Using gridmanagers, proxies, and events in the SPEEDES framework (see Section 4.3.1) all require communications between simulation objects–some of which are located on a physically separate nodes, therefore the efficiency of these built in communication algorithms is characterized.

Characterizing the communications efficiency of the parallel framework depends on not only the communications algorithms in SPEEDES but the configuration of resources such as the following:

- number of nodes used in the simulation

- number of central processing units (CPUs) per node
- communication backplane

Thus in support of this experiment expects to learn the most efficient configuration for a given set of system parameters for a generic SPEEDES application.

Using a new tool is not without pitfalls if one does not understand how the tool works. That is why it is necessary to validate not only the user's understanding of the functionality of the tool, but also the tools limitations, if any, of the particular features being utilized. Two SPEEDES features critical to this research include the use of DDM as a UAV sensor function and external interfaces as data collection programs. Both of these features are validated simultaneously with the experiments that follow.

5.2.1 Measuring the Efficiency of SPEEDES. How can SPEEDES run most efficiently on the available Beowulf system configurations? Beowulf systems can be designed with a wide variety of parameters. The parallel computing system options can include the type of hard disk array, memory capacity (fast/slow), cache sizes on almost every hardware component, operating system, management software/hardware, hard disk interface, communications backplane selection, additional specialized hardware requirements, 32/64 bit processor, processor manufacturer, along with many additional managerial and support related items. A detailed look at these configuration details for a large scale computing application is presented in [25]. The systems used in this experiment were not custom designed for a parallel discrete event simulation application, however they have a few options that can be used to configure the parallel environment at runtime: processors per node, type of backplane, and number of nodes. A total of 2 CPUs are available at each node–thus they share memory without a latency penalty–and are the first configuration parameter: processors per node. Both Fast Ethernet and Myrinet communication backplanes are available and are the second configuration parameter. The number of nodes can vary from 1-32 for the Fast Ethernet backplane and 1-16 for the Myrinet backplane. More details of the specific system configuration are presented at the beginning of the next chapter.

Ideally, a shared memory architecture (on the same motherboard) would prove to be the best process communication solution, but the available Beowulf systems only have shared memory for 2 CPUs. So given that shared memory is not available for $n > 2$, the next ideal is a very fast

backplane with high bandwidth so as to minimize communication latency between processes. This experiment is designed to determine if this ideal is true for the SPEEDES framework.

Also contributing to this experiment is the expected efficiency for more than just a fixed number of UAVs. As the number of UAV simulation objects increase, does the efficiency of the SPEEDES application running on a particular parallel system configuration change as well?

5.2.1.1 Experiment: S1 (SPEEDES-1). From the above parameters one can design a test matrix to measure the efficiency of the process communication algorithms in SPEEDES with a generic application on various parallel configurations. This experiment validates sub-objective 1.4: evaluating the efficiency of the parallelized simulation system. Below are the details.

Parallel Configurations. Table 8 presents the variability when setting up the Beowulf system for running the simulation. This experiment combines each of the available system parameters. For example, the first configuration uses only 1 node with 1 CPU per node–the Ethernet backplane is not used of course. A second configuration uses 2 nodes for the application with 1 processor per node with the Ethernet backplane. A variation of this second configuration is using the Myrinet backplane. Two more variations can be done on this configuration by switching to the Myrinet backplane and also varying the number of processors per node. Using this pattern for the available AFIT Beowulf system results in 17 different system configurations.

SPEEDES application. As mentioned above, each configuration is run with each SPEEDES application. The SPEEDES application used for this experiment is a simple straight-line movement pattern for the UAVs. The number of UAVs varies according to this set: $\{10\,, 20,\ 50,\ 100,\ 500,\ 1000\}$–which is representative of pedagogical, medium, and larger problem sizes. No rollbacks should occur in this application, thus the major reason for variation is communication delays between simulation objects.

The application is a simple simulation in which n UAVs are moving synchronously at each simulation time step. In addition, each UAV simulation object has a DDM subscription to every UAV within a three unit range. This means that if UAV 1 the x,y coordinate pair of $(5, 8)$ that it will detect through the DDM proxy all other UAVs within this square $(5 \pm 3, 8 \pm 3)$. Using DDM in the application requires SPEEDES to create the gridmanagers to implement the DDM function

Table 8 System Configuration Parameters

Configuration	Nodes	Backplane	CPUs Per Node
A	1	E	1
B1	2	E	1
B2	2	E	2
B3	2	M	1
B4	2	M	2
C1	5	E	1
C2	5	E	2
C3	5	M	1
C4	5	M	2
D1	10	E	1
D2	10	E	2
D3	10	M	1
D4	10	M	2
E1	20	E	1
E2	20	E	2
F1	25	E	1
F2	25	E	2

thus utilizing the communications to the fullest extent. The simulation end time is set to 12 time units. That number is chosen because the amount of communication that is occurring between the SPEEDES applications and the number of events in the queue in addition to those that are being processed by the SPEEDES framework are enough to produce an acceptable "steady state" for this research. An external module is connected to record the movement of each UAV–this is the program that produces the '.swh' history file (a binary file recording swarm positions for every time step.)

Statistical Significance. A significant representation of data samples is needed to make any reasonable inferences on the data distribution. The Central Limit Theorem from Statistics requires that the sample size be at least 30 before it can be modeled as a normal distribution [72]. Thus, 30 runs are performed for each of the combinations of available configurations (A-F) and number of UAVs.

Table 9 Experiment S1 Test Matrix

UAVs	Configuration					
	A	**B**	**C**	**D**	**E**	**F**
10	30	30	30	30	-	-
20	30	30	30	30	30	30
50	30	30	30	30	30	30
100	30	30	30	30	30	30
500	30	30	30	30	30	30
1000	30	30	30	30	30	30

Test Matrix. Table 9 shows the matrix that combines the configurations with each UAV count. A configuration letter represents all subsets of that configuration, i.e. 'B' in the table represents B1, B2, B3, and B4 as specified in Table 8.

5.3 Parallel Swarm Algorithm Experiments

5.3.1 Accuracy. What experiments are necessary to understand if fidelity has been maintained or increased for the swarm simulation model? The behavior of a swarm across the various implementations (Microsoft, Linux, SPEEDES) is comparable only by observation of identical behavior. It is known that the exact reproduction of the sequence of moves is not possible even with the same set of parameters due to the random number generation variants in each of the implementations. Also, SPEEDES is not expected to produce the same sequence of moves as the Linux version even though these implementations are using the same random number generators because in a distributed simulation UAVs are their own process and thus what would have been a random number in the sequential Linux algorithm for the 40th swarm member is now the 2nd random number for a local process on the 8th node. The result is that even with exact parameters the movements are anticipated to be different on all three platforms. However, forcing the SPEEDES implementation to behave in a serial fashion like the Linux variant is possible. It is also possible to temporarily remove the random number generation forcing a fixed number instead, thereby producing identical results.

5.3.1.1 Experiment: P1 (Parallel-1). Using principles from Mathematics, one can assume certain conditions true, test for them and then make a generalization based on that data.

This precludes an exhaustive comparison of all possible combinations. This experiment validates sub-objective 1.2: accurate parallelization of the simulation model. Below are the details.

Configuration. A necessary requirement for this test is that the randomness be removed from the swarming algorithm. The programs that are tested include Kadrovach's original swarm movement algorithm (command line version), the author's Linux port, and the SPEEDES version. The random number function that is called throughout the algorithm resides in the file called Formation.cpp. It is defined in the drand() function, so all three implementations are recompiled with this function simply returning a value of 34.03. Thus there are 3 configurations in this experiment.

Application. The application is swarm movement of a defined set of particles using the swarm movement algorithm. The execution of this test must ensure that the parameters used in each test are the same. Those parameters are defined by two input files: params.txt and swarm.dyn along with some command line arguments. The params.txt file describes the swarm algorithm parameters while the swarm.dyn file provides the run time swarm model weights. The files used for this experiment are shown in C.1. The command line used to execute this test for Kadrovach's cline/Linux port is:

```
cline /p swarm.dyn /h P1accuracy_cline.swh /m temp.met

      /i 100 /s 1023 /b yes /n 20
```

which indicates the swarm.dyn parameter input file, P1accuracy_{configuration}.swh output file, simulation length = 100, boundary conditions true, and number of UAVs = 20. Even though command line for the SPEEDES version differs significantly uses the same data files and input arguments.

Test Matrix. From the above discussion it is evident that at least 3 tests need to be run. The remainder of the experiment, however, can be assumed true by the second principle of mathematical induction. That principle is used to prove that a particular formula is the correct

solution. Mathematical induction states that a formula is true for any value of parameter n (for $n \geq c$, where c is some constant) if the following conditions are true [91]:

1. Base Case: formula holds for $n = c$, and
2. Induction Step: If formula holds for $n - 1$, then it holds for n.

In this experiment the well formed formula is a deterministic program which is a formal mathematical expression represented by byte code at its lowest level. The parameter that varies is t, the current simulation time. The condition that needs to hold true is the value of the program at time t on multiple platforms. Thus there are six variants needed for this experiment: 2 for each of the 3 configurations ($t = 1$ and $t = 89$). The first is the base case, the second is the induction step where $n + 1 = 89$.

5.3.2 Efficiency. The efficiency of a program usually refers to the time taken to execute a program as measured by an external clock, as is the case for this test.

5.3.2.1 Experiment P2 (Parallel-2). Accomplishing efficiency testing with a parallelized version of a serial program is straightforward. The simplest test is to compare run times for identical parameters for a given variety of parameters. Speedup as discussed in Section 3.5.4 explains more about the relationship of those run times and so is used in this experiment. This experiment validates sub-objective 1.3: optimization of the parallel simulation implementation.

Configuration. No special circumstances are necessary for this experiment. The following standard conditions apply:

1. Random number generation
2. SPEEDES SpeedesServer and External Modules are activated
3. Using the Breathing Time Warp time management mode
4. Using the Proxy-Based communications management mode

Three efficient configurations result from Experiment S1 (see Section 6.4.1.2) and are used for the tests during this experiment as shown in Table 10.

Table 10 Experiment P2 Test Matrix

UAVs	Configuration			
	A	**C1**	**D1**	**D3**
100	1	30	30	30
500	1	30	30	30
1000	1	30	30	30

Application. The application to test the efficiency of the parallelized version of the swarm program uses a similar application as above only running until a steady-state condition is reached (12 time steps). The number of UAVs used in this application vary also according to the findings in Section 6.4.1.2 indicating three transitions in the SPEEDES framework's performance {100, 500, 1000}.

Test Matrix. Applying the various configurations to the UAV counts mentioned in the application section results in the test matrix shown in Table 10. Statistical runs are still needed because of the inconsistencies when running a SPEEDES framework application (probably due to the TCP connections and higher level proxy connection oriented protocol). Decoding the configurations (A, C1, ...) is shown in Table 8. The entries in the matrix that have 30 statistical runs refer to running the parallel SPEEDES swarming application. The entries that have only 1 run are those running the original single-CPU swarming application.

5.4 Reconnaissance Experiments

Several design parameters are developed for the reconnaissance scenarios as described in Section 3.2. Due to the model availability and time restrictions, not all combinations of parameters as listed in Table 4 are tested. Instead a subset of parameters are evaluated against the measures of success, defined in terms of effectiveness and performance. Many ideas for the experiments in this section are borrowed from a similar research approach in [29]. The available parameter subset includes all three scenarios, a number of enemy targets, swarm size, world dimension, and the area search operation. Statistical analysis is not significant in these experiments because the measurements are not dependent on the execution time of the simulation, but rather its output. The swarming reconnaissance missions are based on seeded random numbers, therefore no matter how

many times the test is conducted, it will always result in the same swarm movements and detection results; it is deterministic for a given random seed value.

5.4.1 Measure of Effectiveness. Measuring how many targets are identified in a given world size for a fixed amount of time across the various UAV counts provides the characteristic effectiveness of the swarming reconnaissance model. Success is granted if this measure reports a 90% identification rate. In order for a UAV to identify a target, it simply must be reported as detected in the output file.

5.4.1.1 Experiment R1 (Recon-1). This experiment validate sub-objective 2.4: performing the simulation of reconnaissance scenarios and evaluating mission effectiveness.

Configuration. The same experiment conditions as mentioned in Section 5.3.2.1 apply here. Because of the focus on effectiveness, this experiment only requires one configuration thus the most efficient as shown in Figure 18: Myrinet, 10 nodes, 1 processor per node (D3).

Application. Additional functionality is embedded in the SPEEDES swarming model that includes target and sensor models. Targets are flagged only once and record which UAV detected its location. Targets remain stationary throughout this experiment and are introduced to the world randomly. A flagged target displays a special report in the output file that is uniquely identified allowing for determining the total number of detected targets within a given swarming reconnaissance run. The scenarios designed in an earlier chapter are modified according to the following limitations: terrain is not varied, enemy threats and poise are not implemented, only the "area" mode of operation is tested. Given the efficiency concerns expressed in previous experiments, the UAV swarm sizes are limited to a usable set: $UAV \in \{20, 50, 100\}$. World dimensions are also scaled back accordingly.

Test Matrix. Resulting from the combination the above constraints is a final test matrix for this experiment shown in Table 11. The total number of runs for this experiment is three.

Table 11 Experiment R1 Test Matrix

Scenario Name	E	Mobility	UAVs	World Dimensions	Mode
Pedagogical	10	S	20	1000x1000	area
Passive	20	S	50	1700x1700	area
Active	40	S	100	2000x2000	area

KEY[E number of enemy targets; S Stationary; M Moving]

5.4.2 Measure of Performance. Two measures of performance defined in Section 3.2.3 represent the reconnaissance mission in terms of accuracy, balance, relevance, and timeliness. Given the constraints of this implementation MOP2 is not measurable–the presence of threats are non-existent. Thus only the first MOP is tested.

5.4.2.1 Experiment R2 (Recon-2). MOP1 addresses how well the detected position reflects the target's actual position. This experiment validate sub-objectives 2.4: performing the simulation of reconnaissance scenarios and evaluating mission effectiveness.

Configuration. The same experiment conditions as mentioned in Section 5.3.2.1 apply here. Similar to the previous experiment, the focus on positional accuracy only requires one configuration so the most efficient is selected (D3).

Application. Functionality is embedded in the SPEEDES swarming model to provide time-tagged target position information as well as detected time-tagged target position information. Targets are flagged multiple times as each UAV detects its location. Similar to the above experiment, the targets are randomly placed in the world and then incrementally move slowly in a linear direction. Except for what has already been mentioned the scenario is unchanged from experiment R1. Also the same UAV swarm sizes are limited to an efficient set: $UAV \in \{20, 50, 100\}$.

Test Matrix. The resulting test matrix is shown in Table 12. Thirty samples are enough to perform this experiment.

Table 12 Experiment R2 Test Matrix

Scenario Name	E	Mobility	UAVs	World Dimensions	Mode
Passive	20	M	50	1400x1500	area

KEY[E number of enemy targets; S Stationary; M Moving]

5.5 Summary

The experiments for this research are categorized into three categories: PDES experiments, parallel swarm algorithm experiments, and reconnaissance experiments. All experiments are tied to objectives as stated in the beginning of this document. An experiment is distinguished by its purpose, configuration, application, and test matrix. The next chapter documents performing the tests described in this chapter.

6. Testing & Analysis

6.1 Introduction

Testing and Analysis comprises the actual tests as performed and subsequent analysis of the resulting data. A test is an executed experiment. All experiments are described in Chapter 5. First the characteristics of the systems upon which the experiments are conducted are presented. Second the important use of scripting is discussed. Then according to the order of the experiments the data collection and analysis is presented.

6.2 High Performance Computer Systems

What systems are used, and what are the implications of each one? Because this research implements a parallel simulation it is imperative to use a parallel architected computer system. This system is commonly referred as a Beowulf system and is introduced in Section 2.5.4. While several different Beowulf configurations exist at AFIT, they can be roughly categorized by performance into two levels: Fast Ethernet [23] backplanes and Myrinet [76] backplane. Both categories allow for parallel computing but Myrinet has a higher performance (greater throughput, lower latency). Thus only one system for each category is used for this experiment: Aspen (Fast Ethernet) and Aspen (Myrinet). Yes, Aspen has both types of backplanes on one the same cluster. Aspen is the name given to the AFIT Beowulf cluster (it is also the name of the manufacturer, see [9]) and it's configuration is summarized in Table 15. It contains both categories across a total of 48 nodes with two processors per node, the major difference being that only a subset of Aspen has the Myrinet backplane connection. Appendix A describes the processing control structure, memory architecture, interconnection network, and scheduling software for AFIT's HPC systems.

6.3 Scripting

When conducting hundreds of runs it is necessary to understand how to write a script to perform the experiments in an automated fashion. Because the Beowulf cluster is Linux-based, the operating system has built-in scripting support through shells. All scripts used with this research are based on the bash shell interpreter because of the ease of use and excellent documentation available in [24].

In addition to the hundreds of runs, there is another motivation for using scripts with SPEEDES applications. Often it is necessary to startup a SpeedesServer application (the communications server) before any non-local (on a physically different node) simulation objects can interface to the simulation. In addition, use of the External Module feature also requires use of the SpeedesServer and itself is a separate application. Finally, whenever a simulation is run on a Beowulf architecture all participating nodes must have their own instance of the SPEEDES application running locally on their machine–unlike the automated task distribution in a message passing interface parallel application. This means that if 10 nodes are participating, then every time the one simulation run begins, the user must log into each one of those nodes and a copy of the application must be started. A program called Spexec automated this remote login process. It was provided by the Air Force Research Laboratory/IFTC, advanced computing architectures branch, out of Rome, New York. Thus at any one time there are at least 3 separate programs and at most $3 + n$ separate programs potentially running on different machines. Use of scripts to automate this process relieves the administrative burden of starting and quiting all associated SPEEDES applications for a single simulation.

6.4 Parallel Discrete Event Simulation Experiments

Testing phenomena and results are analyzed for the parallel discrete event simulation experiments.

6.4.1 Experiment S1.

6.4.1.1 Testing. Scripts ran all of the runs for this experiment sending jobs to the PBS queue and collecting the data in descriptive named files. During testing, several times, one of the jobs would hang in the "run" status at which point it was discovered that in the middle of the statistical runs for a particular job one of two situations would occur as described below:

1. Spexec would not correctly spawn all required processes, so that for a $m > 1$ node simulation, the SpeedesServer would not allow the simulation to begin because at least one simulation object (n) has not yet broadcasted its existence. So there was an anomaly with Spexec correctly launching all n processes for a simulation of n UAVs on m processors.

2. The SpeedesServer would never get past its initialization sequence because not all simulation objects could connect. The error message contained these words, “Couldn’t bind main socket stream.” This was associated with some sort of timeout that a SPEEDES expert would understand.

In both of these situations, the run was aborted and counted as a statistical loss.

The plots shown in the analysis section might have missing data points for the above reasons. Recall that the limitation on the Myrinet system is at 16 nodes, therefore any runs that require more than 16 nodes do not have any associated data. Also, the base case of running on 1 node need only be run 1 time, thus the remaining configuration variables that vary the processor per node count and the type of backplane are not factors for the 1 node runs, thus these configurations in the plot below do not have any associated data.

6.4.1.2 Analysis. The metric of interest concerning efficiency is elapsed time. SPEEDES has a default output field that provides the total wall time used to indicate the length of the entire simulation. That field is specified by “wall=” in the standard output of the end of a SPEEDES application. In parallel computing technology this metric is the equivalent of wall clock time.

Since 30 runs were performed with each of the entries in the test matrix (see Table 9) a box plot analysis is conducted to show the sample median, the interquartile range (middle 50% range of the data), and any outliers of the execution time for the given parameter sets. Figure 11 shows this data for 1000 UAVs. See Appendix C for box plots of all data for Experiment 1. What additional insight does this box plot give? The 3 boxed numbers written parallel to the y-axis indicate actual values of the median in the nearby box revealing similar execution time at the lowest levels. No one configuration stands out above the rest when comparing median values. It does not make sense to perform speedup calculations on the SPEEDES program itself with varying numbers of UAVs, although that information is available. One might ask about the outliers down near the ’0’ value. After probing the data, these runs were not true runs because of the reasons discussed in Section 6.4.1.1.

A more meaningful plot is shown in Figure 12. It contains the same format of the configurations across the bottom as in the figure above, but this time in addition to the base UAV count

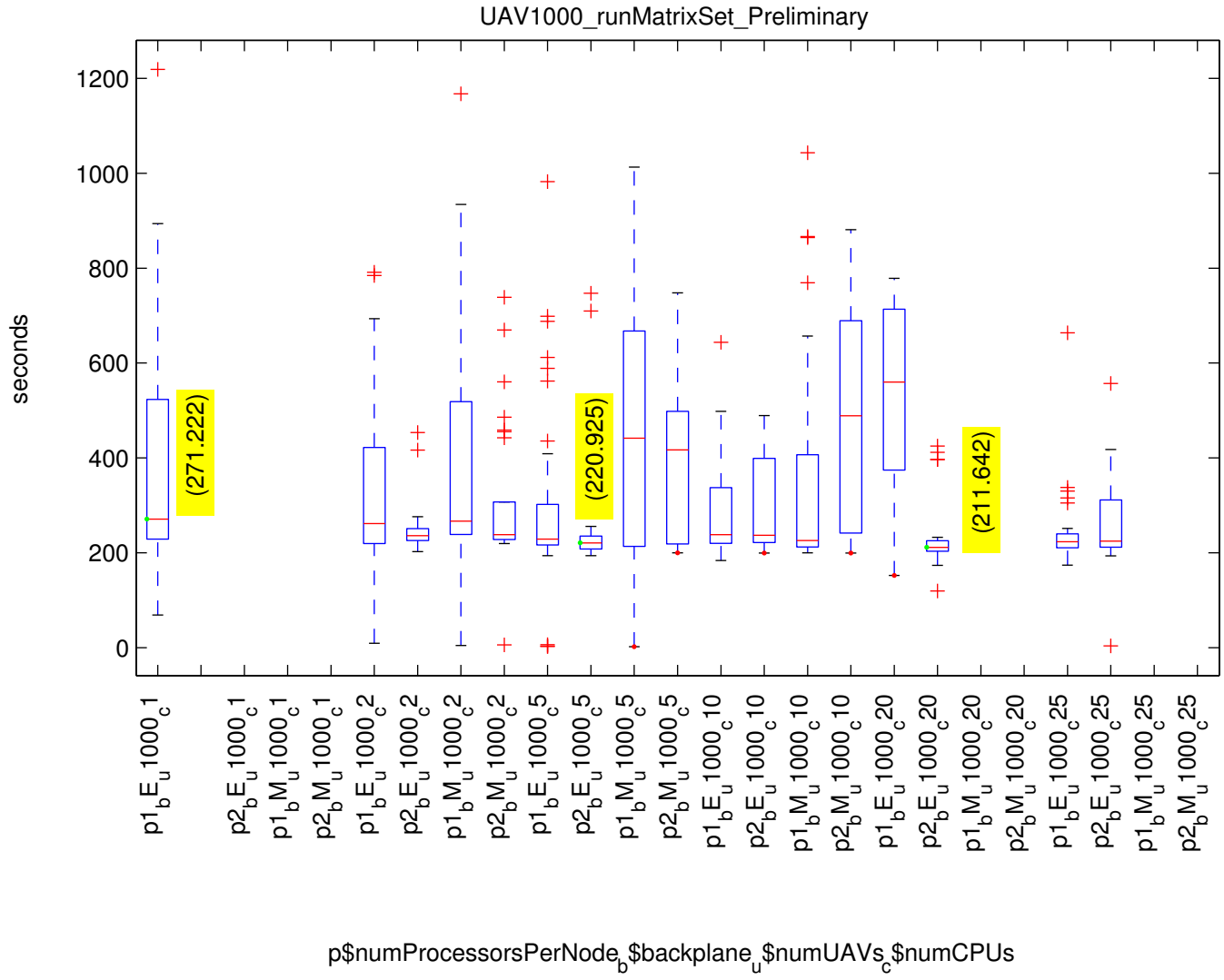


Figure 11 Box plot of Elapsed Time Data for Experiment 1 for 1000 UAVs

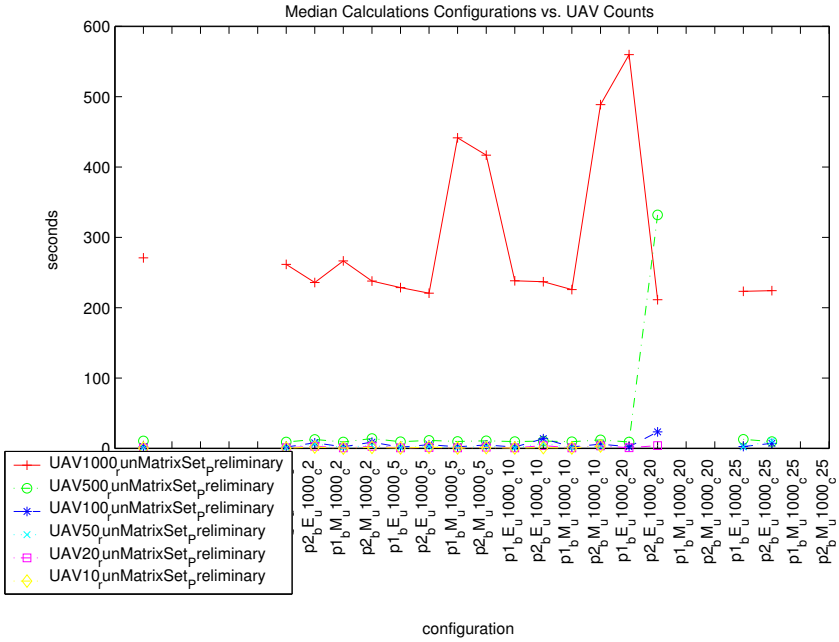


Figure 12 Median Values of Configurations vs. UAV counts

(1000) for the figure, all of the remaining UAV counts are laid over this plot to contrast the various execution medians. The data points are the median values of the 30 statistical runs, identical to the solid line in the middle of the box in the box plot figure. This figure shows a major demarcation for the SPEEDES framework when transitioning from hundreds to thousands of UAV objects. Thus it can be expected that simulations involving over 1000 simulation objects that require intercommunication will take at least 200 seconds to execute 12 simulation time units.

Probing further into the plot reveals another solid demarcation at 500 UAVs, however at 100 UAVs and below the efficiency of the SPEEDES framework is relatively close ($< 5\,seconds$).

The final analysis of interest is finding a subset of the configurations which are consistently more efficient. Both Figures 13 and 14 reveal there are certain configurations which always take longer no matter how many UAVs. The configurations with 2 processors per node are eliminated. The remaining values are easily rank ordered by the average normalized median values. The top three configurations across all remaining configurations and UAV counts are shown in Figure 15.

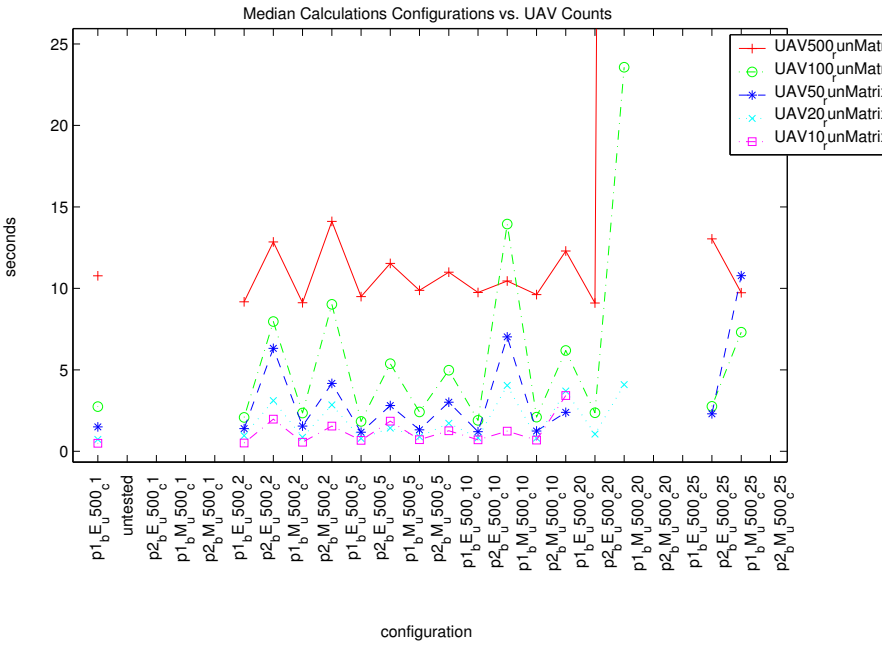


Figure 13 Median Values of Configurations vs. UAV counts (up to 500)

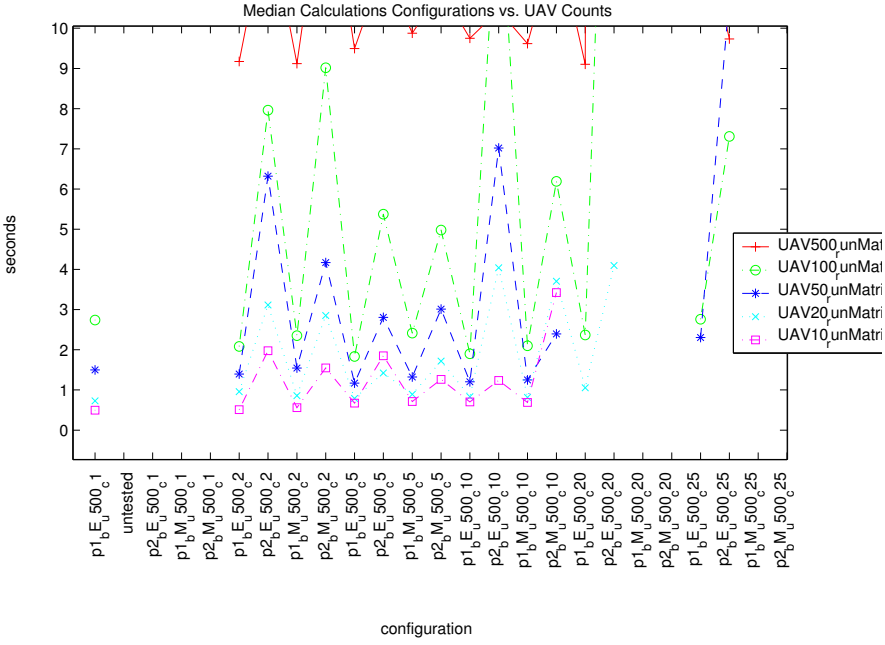


Figure 14 Median Values of Configurations vs. UAV counts (up to 100)

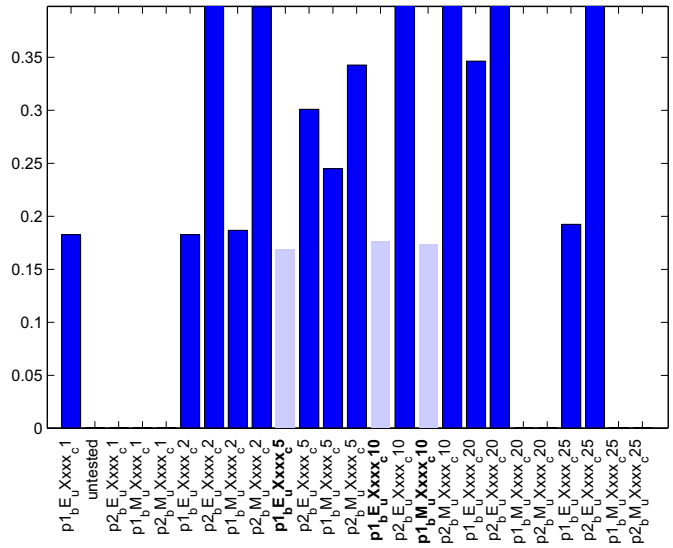


Figure 15 Average Normalized Median Values for Each Configurations

6.5 Parallel Swarm Experiments

These results and analysis pertain to the parallel implementation of the swarming algorithm.

6.5.1 Experiment P1.

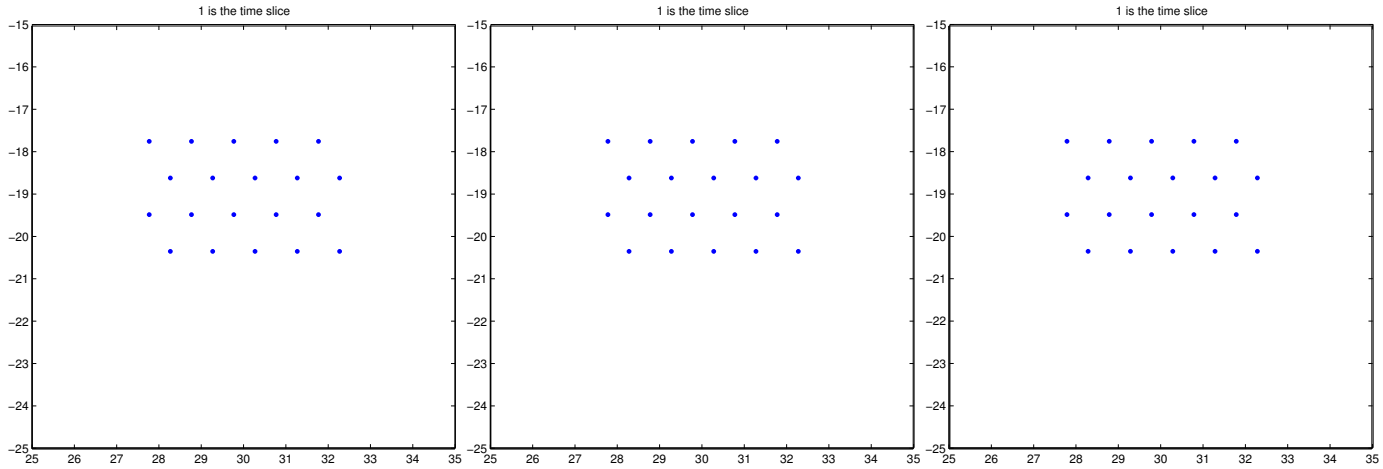
Refer to Section 5.3.1.1 for design details about this experiment.

Running the experiments with varying values of t results in a binary output file that must be viewed through a visualization program. Thus the results are evaluated empirically. Execution of the test is simply using the correct parameter files and command line arguments. The output is captured for all six runs and shown in Figures 16 and 17. By observation, the output is identical for both cases of the induction procedure, therefore it is proven that these algorithms behave accurately for all values of t.

6.5.2 Experiment P2.

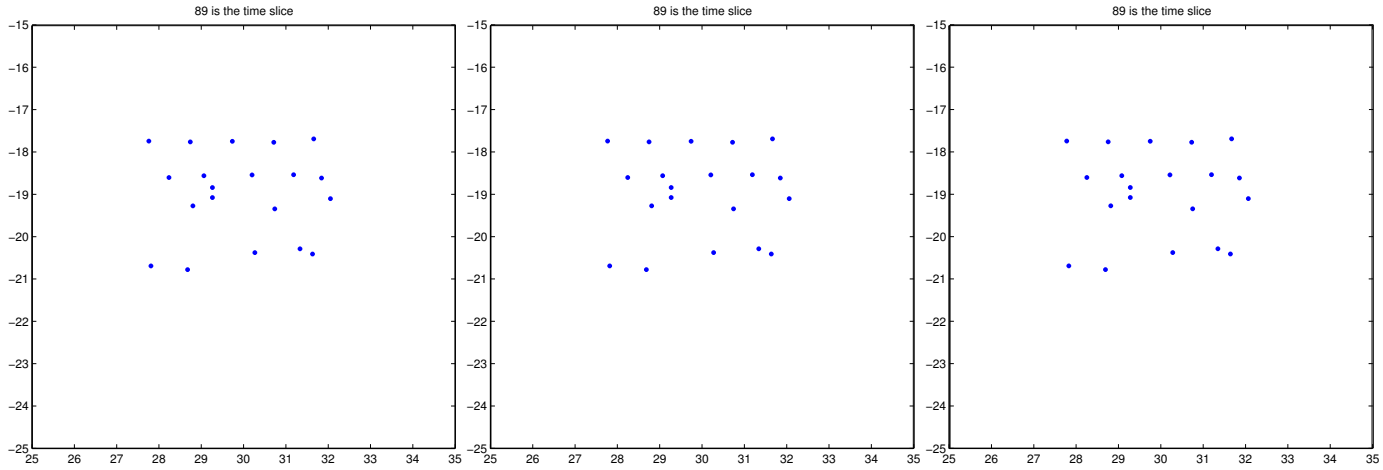
Refer to Section 5.3.2.1 for design details about this experiment.

Data is collected in like manner to Experiment S1 using script files to execute, gather, and process the test data. It did not take long before it was evident that there were some efficiency problems. The first row in the test matrix was not the issue, however when the runs from the second row started executing, hours had passed and the SPEEDES run with 500 UAVs was still not complete even though the original serial implementation had finished within the first 5 minutes of execution. After probing the output files it was clear that a number of inefficiencies were discovered. Those



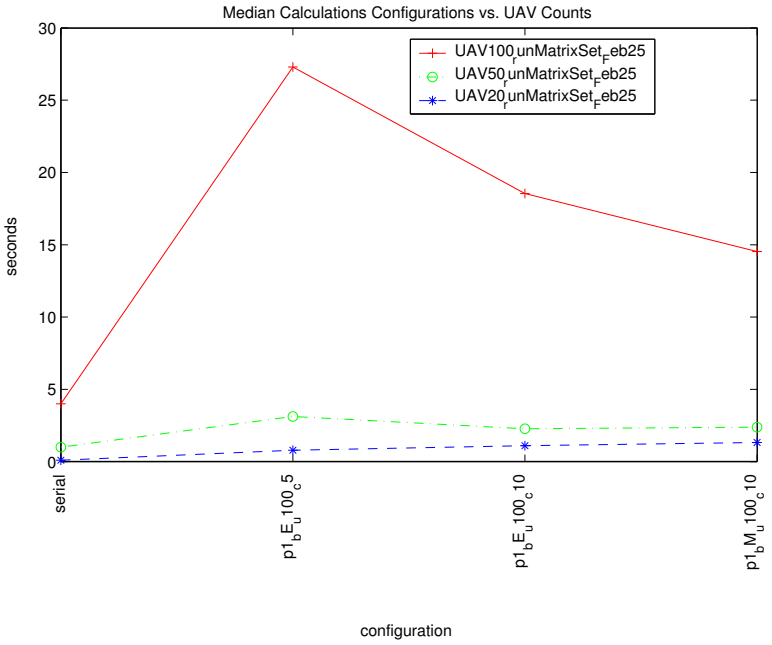
(a) Kadrovach's cline (b) Linux port (c) SPEEDES

Figure 16 Experiment P1 at $t = 1$



(a) Kadrovach's cline (b) Linux port (c) SPEEDES

Figure 17 Experiment P1 at $t = 89$



(a) SPEEDES

Figure 18 Experiment P2 Median SPEEDES Execution for UAVs$\in \{20, 50, 100\}$ vs. Serial Execution

items include the number of proxy connections, number of rollback events, and the number of events needing processed before simulation time was allowed to advance. At that point it was clear that the working parallel swarm SPEEDES implementation was not efficient enough for running larger UAV counts (>100). Further investigation is conducted in the *Analysis* paragraph below.

The test was modified then to only include smaller UAV counts (<100) and rerun. The output is shown in Figure 18. As expected, the difference in execution times is negligible for values of 50 and under. Unexpected, however, is the lack of speedup for the largest data set. Speedup measures how much quicker the parallel implementation executes the program over the serial, but as seen in this chart any speedup calculation is less than 1, indicating no speedup at all. The larger data runs did finish on the serial version and the execution times are presented in Table 13.

Analysis. An intrusive look into the SPEEDES framework is required for understanding the reason for this inefficiency. A preliminary investigation modified configurable SPEEDES implementation options, re-executed the run, and found the following:

Table 13 Experiment P2 Serial Execution times for UAVs $\in \{100, 500, 1000\}$

UAV Count	Execution Time (seconds)
100	4
500	198
1000	1478

- original unsuccessful run
  - last output line after 12512 seconds of execution; average 1042 wall clock seconds/1 simulation second

```
2161) GVT=12 cpu=92747.2 wall=12512.9 STAR=0
Tproc=73722.5 Tcmt=0 Eff=0 Nproc=862 Ncmt=10042
e=3310020 e/c=153 eg=1290338 r=1632526 m=54009 a=0 c=0
```

  - KEY for select values of the above SPEEDES output line
    - GVT=latest simulation time that has been committed (cannot be rolled back)
    - wall=total wall time spent since the start of the simulation
    - e=total number of events processed since the start of the simulation
    - eg=total events committed since the start of the most recent GVT cycle
    - r=number of rollbacks since the start of the simulation
    - m=number of event messages sent since the start of the simulation
    - a=number of anti-messages sent since the start of the simulation

- c=number of events canceled since the start of the simulation

- 2nd attempt
- CHANGE: basic SPEEDES algorithm from Breathing Time Warp to Breathing Time Buckets
- THEORY: Breathing Time Buckets will always process at least 1 event so no chance for deadlock to occur
  - last output line after 11590 seconds of execution; average 966/1

```
2799) GVT=12 cpu=97723 wall=11590.7 STAR=0 Tproc=73695.2
Tcmt=0 Eff=0 Nproc=28 Ncmt=1858 e=3310020 e/c=118
eg=450691 r=2907390 m=54009 a=0 c=0 1073
```

- 3rd attempt
- CHANGE: number of processors: 5 CPUs
- THEORY: Communication is causing the simulation delay, therefore with half the CPUs there is much less needed communication
  - last output line after 22237 seconds of execution; average 1853/1

```
5016) GVT=12 cpu=83218.1 wall=22237.5 STAR=0
Tproc=75895.9 Tcmt=0 Eff=0 Nproc=61 Ncmt=362 e=3280515
e/c=131 eg=1267218 r=5543373 m=24008 a=0 c=0 1373
```

From these limited attempts there is no obvious solution to improving the efficiency of processing the of the discrete events for this swarming application. Obvious eyesores are the millions of rollbacks that are occurring on a regular basis. This is probably due to the optimistic processing of future events that need rolled back because each UAV depends on the data that has already been modified at a future simulation time, so that n UAVs are causing rollbacks on all other UAVs. This potentially could lead to thrashing and even deadlock–but that is obviously not the case here. Other

Table 14 Boundary Scaling

Scenario	**World Dimension**	**Scaled**	**Repulsion Region Scale**	**Scaled World Dimension**
Pedagogical	1000x1000	20x20	10x10	500x500
Passive	1300x1300	26x26	16x16	800x800
Active	1600x1600	32x32	22x22	1100x1100

attempts were made by changing the update interval so that each UAV has a designated time slot offset by $1/numUAVs$ but this took longer even though it had 0 rollbacks! One other possibility is that the performance of SPEEDES gets worse before it gets better, so that at even higher data loads, swarming SPEEDES outperforms the serial version. This is certainly true for data sets that run out of available memory on 1 CPU.

6.6 Reconnaissance Experiments

Initial efforts demonstrate the capability of a swarm to perform a reconnaissance mission as shown in Figure 19. The "plus" symbols indicate fixed targets, the dots represent the pursuing swarm. The displayed data is the result of doing one run of the simulation. The swarm moves freely about the search space for a fixed period of time (typically 1800 seconds simulation time) scanning for targets while it is steered as a result of the pseudo-random interactions of the emergent behavior.

Executing reconnaissance testing required more parameterization than what is developed in the design of experiments. Common to both experiments discussed below are several conditions that occurred during testing which highlighted the importance of selecting the correct an appropriate boundary region, sensor footprint, and simulation duration. One of the largest struggles, determining the boundary box, is due to the boundaries being built-in to the swarm algorithm. Using Kadrovach's implementation brings on 4 different ways of setting parameters that control the behavior of the swarm–one of which is the boundary area. Overcoming this struggle was accomplished by understanding how the `scale` parameter influenced the boundary conditions which in-turn influences the swarm as a whole. Table 14 shows this relationship. The "World Dimension" column is the number entered into the params.txt file that is read by the simulation. The

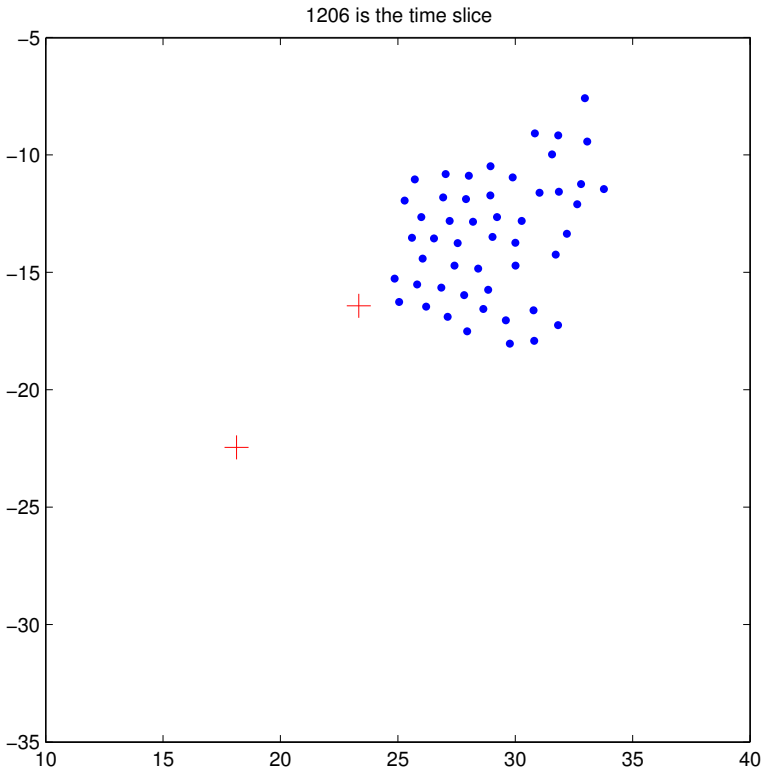


Figure 19 Reconnaissance Demonstration

"Scaled" column accounts for the `scale` factor associated with this simulation (50 for this test). Thus 1000/50 = 20. Scaling is pervasive throughout the swarm algorithm and so even the boundaries are scaled down. The next column accounts for the 'padding' that is inherent to the swarm algorithm which allows the swarm to repulse strongly from the boundary. Effectively, it is a 5 unit padding on all edges prior to the bounded edge in which on occasion a swarm member might enter, but most often does not. Thus a 20x20 area reduces to 10x10 because of the 5 units of padding on the East and West sides and North and South edges. The final column is the actual area in which the swarm can move scaled back to the original units as the world dimension (multiply column 3 by `scale`).) The key to understanding the boundary box is that targets must exist within that box with the proper units (1000's or 10's) when placing them in the simulation. Simply using the default (2000x3000) resulted in poor effectiveness because the area was outside that of the range of the swarm during the reconnaissance mission.

Testing a reconnaissance scenario required the definition of a sensor footprint for each UAV. The sensor functionality uses the DDM feature of SPEEDES, specifically the double range filtering part. To maintain consistency with the normalization (see chapter 4 of Kadrovach's dissertation [54]) of the swarm positions, the sensor footprint is set to a diameter of 1.3. Thus in a square centered about the current UAV position the footprint extends 0.65 North, South, East, and West. The extra 30% is included for sensor overlap between members of the swarm to produce a continuous coverage area when in a strict lattice formation. It is assumed the sensor footprint of the swarm is 20% of the scaled world dimension for these runs. For example, suppose for Pedagogical example there are 20 UAVs, thus a 20 unit sensor foot print in a total of 100 unit area, thus 20/100 = 20%. Maintaining this ratio is the key to success.

The simulation duration is chosen conservatively. Finding current threats in a timely manner depends on the current concept of operations for the user of the swarming reconnaissance. All enemies can be found given enough time and resources, thus it is assumed that the tests in this resarch are restricted to a window of 1800 seconds (30 minutes).

6.6.1 Experiment R1.

6.6.1.1 Testing. Running this test required the ability to validate the function of reconnaissance as performed by the swarm. In particular, it is essential to know, given a UAV's path through the bounding box how many targets were identified. In order to validate that a given UAV is properly 'identifying' targets, a visualization is used. The output from the test (SPEEDES data) is compared to the swarm history file overlaid on the target map. This method is demonstrated in Figure 20. The targets are shown as "plus" symbols, the swarm is annotated as "dots." Thus when a swarm member comes near enough to a target its sensor should detect it. To visualize the sensor footprint, a box with the exact dimensions of the sensor is drawn around each UAV. Thus it is easily seen when one UAV should detect a target. For validation purposes the current position of the UAV as well as the target during each simulation time step is available.

In an effort to see how each random number seed affected the resulting search pattern, there was a different source of randomness discovered. Running a reconnaissance scenario on a set of 10 CPUs with a given random number seed generates a different search pattern each time it is run–while not changing any inputs! This is attributed to the connection oriented PDES. When running the simulation, each node acts as 1 or more UAVs and thus must communicate any updates to the SpeedesServer. The algorihm is such that each UAV schedules an update simulataneously every 1 simulation second. Therefore, depending on who makes the connection first with the SpeedesServer is the order in which the events are logged. That update requires a call be made to the SpRandom object by all UAVs. This random number generator allows for distributed consistency in that it produces the same pseudo random number sequence for a given seed value. Why is each UAV update generating different positions with the same seed value? Because the order in which the SpRandom function is called differs based on the pecking order of the UAV nodes that are simultaneously executing the move_update function.

Thus for one simulation, UAV 0 might call SpRandom first, second, first, fourth, and tenth because that was the order in which the call was presented to the SPEEDES management. The next time the simulation is run, that order varies based on which CPU finishes its process first. The other randomizing issue that is complicating matters is the idea of rollbacks. In all these simulations, millions of rollbacks are occurring, thus if 10 UAVs are posting updates at t=1 and are rolledback, the original sequence of calls to the SpRandom function is not preserved, because there is no strict ordering of UAV updates inside an update interval. The serial (event-based)

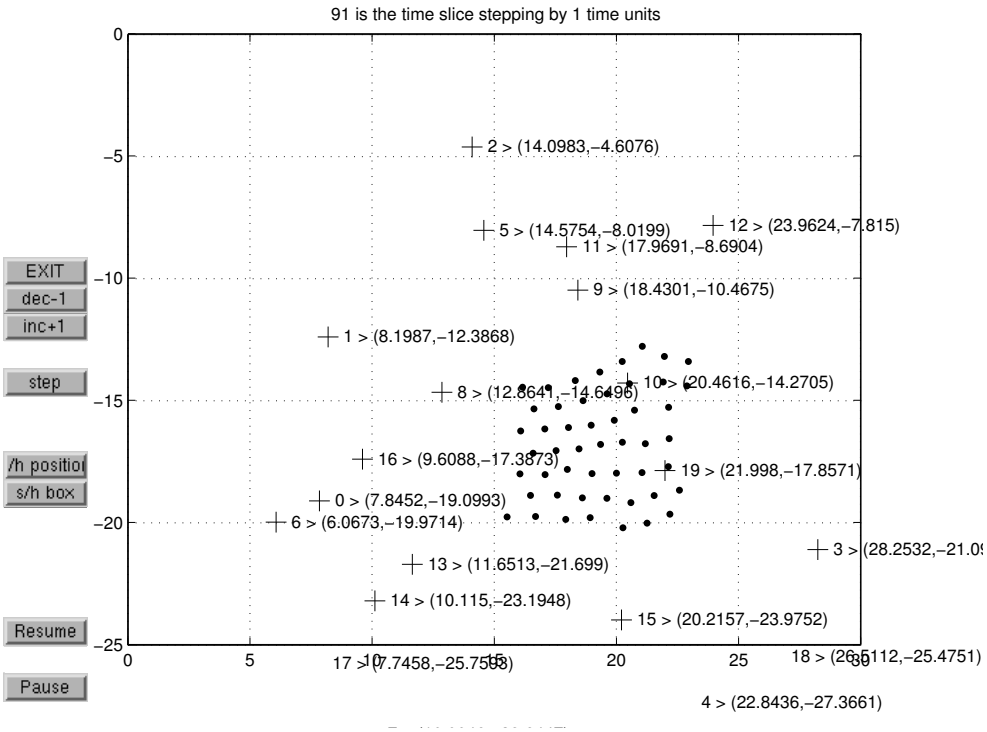


Figure 20 Visualization Symbology

parallel implementation does just that, so it remains predictable even with rollbacks, because the inbetween calls are ordered in time and must therefore be kept coherent through a rollback.

6.6.1.2 Analysis. To further research this randomness phenomena it is postulated that the backplane communications has a major influence on the ordering of calls to the SpRandom function, thus 6 identical tests were ran on one CPU and the binary swarm history file compared with the Linux command, "cmp -l file1 file2." This preliminary test resulted in 5 of the 6 files being identical!! Thus it corroborates the theory that the inter-process communications introduce randomness to this simulation! Further research should be done to examine the appicabilty of this random side affect to security algorithms. It is incorrect to maintain as stated in Section 6.6 that for a given random seed value the results of the program are deterministic.

Nonetheless, this should not impact how many runs are accomplished because there is an expectation of randomness anyway, thus no extra bias is introduced. Due to the experiment focusing on effectiveness rather than the execution time of the SPEEDES program, only 5 runs are accomplished for each scenario. A typical run-through consists of validating postion information

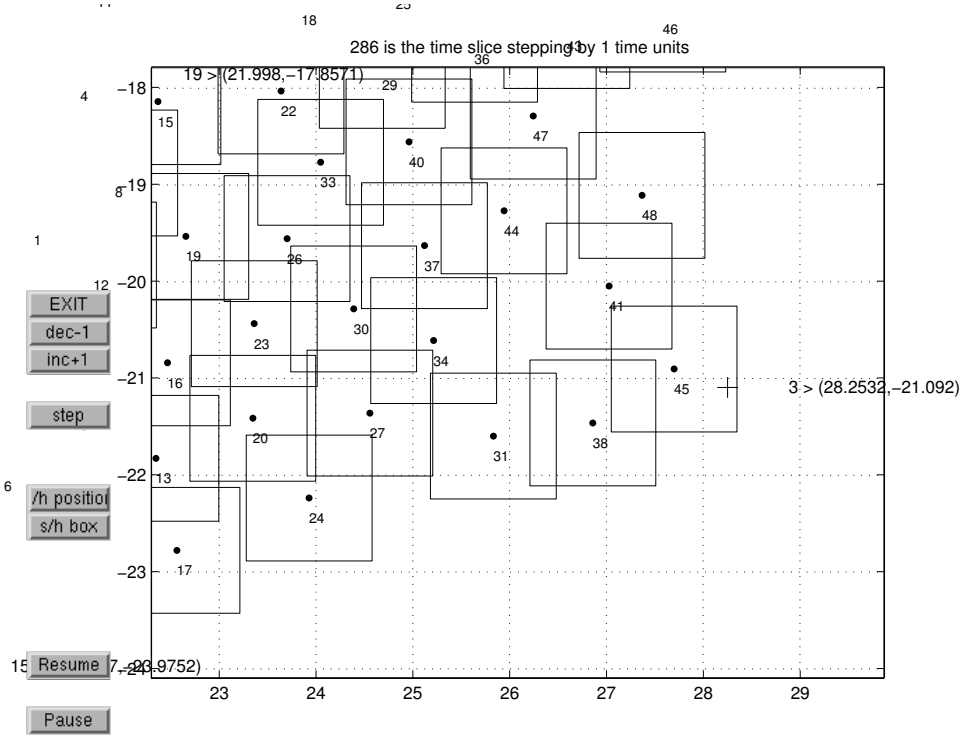


Figure 21 Reconnaissance Visualization

with both the visualization and the SPEEDES output. Figure 22 shows a typical SPEEDES output (filtered for target dectections). The target number is first followed by the UAV that detected it, the time, and the target position. This is cross-referenced to Figure 21 at timestep 286 which is right after UAV 45 detected target 3–which agrees. Fig 21 includes the box and number identifying the sensor range. One minor issue is the transition as shown in the visualization from the non-detect to detect state. According to Fig 22 the sensor box surrounding UAV 45 should encompass target 3 at exactly time slice 284. This does not happen, rather the box crosses a little too soon or not soon enough. This is probably a function of the visualization implementation.

A summary of the results for all three averages is shown in Figure 4. These favorable results indicate the emergent behavior is random enough to cause the sensor footprint of the swarm to cover the percentage of the the enemy territory such that at least 90% of the targets are identified. This measure of effectiveness is dependent on the parameters previously discussed and therefore should not be considered apart from them.

```
8 target_detected by UAV 1 at 1.3 target position is ( 12.8641, -14.6496 )
10 target_detected by UAV 46 at 1.3 target position is ( 20.4616, -14.2705 )
19 target_detected by UAV 48 at 51.3 target position is ( 21.998, -17.8571 )
3 target_detected by UAV 45 at 284.3 target position is ( 28.2532, -21.092 )
12 target_detected by UAV 28 at 696.3 target position is ( 23.9624, -7.81496 )
9 target_detected by UAV 4 at 767.3 target position is ( 18.4301, -10.4675 )
11 target_detected by UAV 11 at 777.3 target position is ( 17.9691, -8.69035 )
5 target_detected by UAV 11 at 883.3 target position is ( 14.5754, -8.01987 )
16 target_detected by UAV 3 at 974.3 target position is ( 9.60883, -17.3873 )
0 target_detected by UAV 3 at 1044.3 target position is ( 7.84519, -19.0993 )
6 target_detected by UAV 3 at 1120.3 target position is ( 6.06726, -19.9714 )
1 target_detected by UAV 4 at 1234.3 target position is ( 8.19865, -12.3868 )
2 target_detected by UAV 46 at 1770.3 target position is ( 14.0983, -4.60758 )
```

Figure 22 Sample SPEEDES output

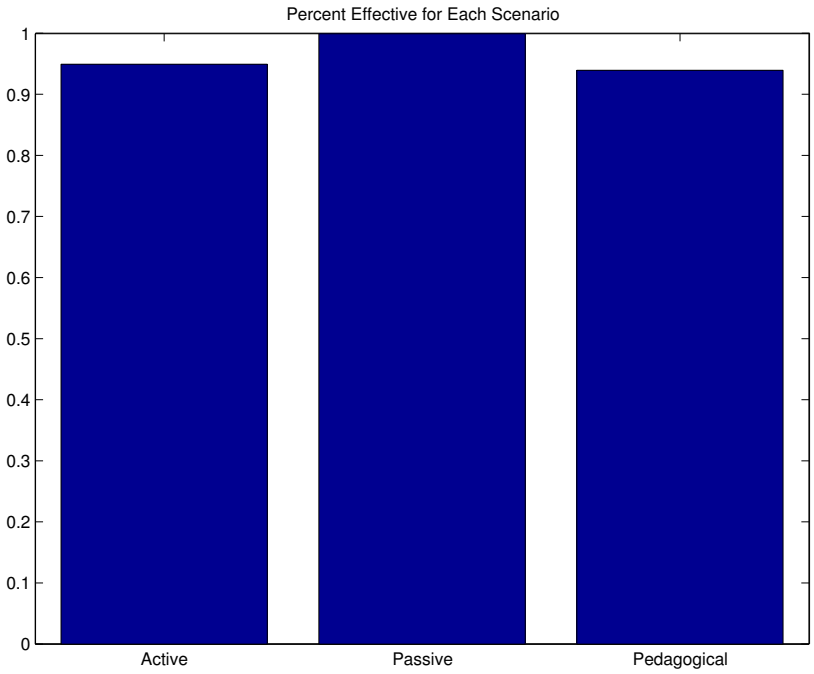


Figure 23 Experiment R1 Results

6.6.2 *Experiment R2.*

6.6.2.1 Testing. Because this is measuring the accuracy of tracking a moving target, it is not necessary to simulate an entire recon mission. Instead, enough data points that provide a statistical sample from one run is sufficient. Figure 24 shows this data. In it are the UAVs which were tracking targets that move throughout the course of the simulation. It reads from left to right, target 1 detected by UAV 0 at time 4.3 with xloc,yloc and actually did have xloc,yloc available. It shows the targets moving within a simulation, but still remaining in the search/sensor range. This run is done with 50 UAVs and 20 moving targets. Only 25 seconds are simulated–which is plenty of data to analyze.

6.6.2.2 Analysis. Using the SPEEDES framework to implement the sensor model has many perks. One of which is obvious by the results shown in this section. With the SPEEDES framework, the accuracy is 100%. Should it be necessary to integrate a differing sensor model, this test can be used to validate the model's vulnerabilities or accuracy benefits.

6.7 *Summary*

This chapter informs the reader of the high performance computing systems upon which the tests perform functions that are analyzed. The importance of scripting is mentioned as it is a key benefit toward customization, automation, and execution of testing runs. The experiments as described in the previous chapter present data and results that had an element of surprise. Through efficiency analysis several configurations of the available computer systems suffered elimination. The remaining configurations served as a basis for parallel swarming testing and initial reconnaissance exploration. The latter remains incomplete. Next, the final chapter of this research effort includes comments and conclusions about the research conducted.

target	detectedby	time	xloc,yloc	xact,yact
1	0	4.3	9.90676, -9.80623	9.90676,-9.80623
1	0	5.3	10.6568, -9.80623	10.6568,-9.80623
1	11	6.3	11.4068, -9.80623	11.4068, -9.80623
1	11	7.3	12.1568, -9.80623	12.1568, -9.80623
1	14	8.3	12.9068, -9.80623	12.9068, -9.80623
1	21	9.3	13.6568, -9.80623	13.6568, -9.80623
1	25	10.3	14.4068, -9.80623	14.4068, -9.80623
1	28	11.3	15.1568, -9.80623	15.1568, -9.80623
1	35	12.3	15.9068, -9.80623	15.9068, -9.80623
1	39	13.3	16.6568, -9.80623	16.6568, -9.80623
1	46	14.3	17.4068, -9.80623	17.4068, -9.80623
1	49	16.3	18.9068, -9.80623	18.9068, -9.80623
4	49	16.3	17.0455, -8.17654	17.0455, -8.17654
8	0	3.3	10.1367, -9.12277	10.1367, -9.12277
8	0	4.3	10.1367, -8.37277	10.1367, -8.37277
8	1	2.3	10.1367, -9.87277	10.1367, -9.87277
8	0	4.3	10.1367, -8.37277	10.1367, -8.37277
8	1	2.3	10.1367, -9.87277	10.1367, -9.87277
8	1	2.3	10.1367, -9.87277	10.1367, -9.87277
8	4	3.3	10.1367, -9.12277	10.1367, -9.12277
8	7	4.3	10.1367, -8.37277	10.1367, -8.37277
8	8	1.3	10.1367, -10.6228	10.1367, -10.6228
8	8	2.3	10.1367, -9.87277	10.1367, -9.87277
8	8	2.3	10.1367, -9.87277	10.1367, -9.87277
8	8	2.3	10.1367, -9.87277	10.1367, -9.87277
10	39	2.3	15.3965, -9.61033	15.3965, -9.61033
10	42	3.3	15.3965, -8.86033	15.3965, -8.86033
10	42	4.3	15.3965, -8.11033	15.3965, -8.11033
10	43	1.3	15.3965, -10.3603	15.3965, -10.3603
10	43	2.3	15.3965, -9.61033	15.3965, -9.61033
19	48	1.3	17.2102, -13.5934	17.2102, -13.5934
19	48	2.3	17.9602, -13.5934	17.9602, -13.5934

Figure 24 Experiment R2 Comparison

7. Conclusions

7.1 Introduction

The previous chapter provides the quantitative support so that a qualitative assessment can be made about progress in this research effort. As stated in the Chapter 1, the research focus centered around addressing the problem of future requirements driven by current military affairs. Those new requirements include safer, more accurate, fault-tolerant weapon systems. The goal was understanding how emerging technologies involving UAVs can be used to satisfy those future military requirements. The strategy was through a swarming reconnaissance simulation. This chapter indicates the level of success or failure in achieving that goal. The objectives are restated and conclusions drawn about each lower level objective based on the quantitative data in Chapter 6.

7.2 Completion of Objectives

Two high level objectives were defined in Chapter 1 as one method pursuant to the stated goal of this research. Below each listed objective are the sub-objectives that further define specific ways to mark success or failure associated with each objective.

7.2.1 Develop, parallelize, and evaluate a swarm model simulation system .

Development and validation of a swarm simulation model. This model is developed using the SPEEDES framework by integrating existing swarm models from the swarm research community. Chapter 6 Section 6.5.1 indicates this sub-objective is complete.

Accurate parallelization of the simulation model. Quantitative evidence from the experiments conducted in this research indicate that the parallelized swarm simulation model has at least the same accuracy as the original serial version. From the evidence in Chapter 6 Section 6.5.1 it can be concluded then that success was achieved for this sub-objective.

Optimization of a parallel simulation implementation (efficiency). The optimization of parallel simulation is inherent to the SPEEDES environment thus it provides for many

optimizations (Breathing Time Warp, for example). Additionally, optimizations of the parallel implementation of the swarm system can be done explicitly through design changes at the algorithm level. This is accomplished by changing the design from an event-driven to a proxy-driven implementation. While not mentioned at the implementation level, several additional optimizations concerning the position update portion of the algorithm are accomplished. Specifically, the angular update approach decreases to a minimum the communications between swarm members. A second optimization, uses the implicit DDM function of the framework to reduce the calculation overhead of the main `findneighbors` portion of the swarm algorithm. While not all of these optimizations are implemented, they provide the concepts needed to implement an even more efficient parallel simulation. Optimizations are included in the current parallel implementation, thus this sub-objective is complete.

Evaluation of the efficiency of the parallelized simulation system. In a straightforward manner the testing and analysis portion of this document recorded the efficiency of the parallel system. This is simply a measure of elapsed time as plotted against varying inputs and configurations. See Chapter 5 for details. The results of the evaluation in Chapter 6 Section 6.5.2 indicate the parallelized simulation is much slower than the serial version for larger problem sizes. Therefore, it can be concluded that this sub-objective has been completed.

7.2.2 Evaluate effectiveness of a swarming reconnaissance mission.

Development of supporting models. While many supporting models are discussed during the design, only an essential subset were implemented: high-fidelity swarm behavior, low-fidelity sensor, and targets, and communications. The swarm model is the successful parallelized version mentioned in the previous objective. The remaining support models are all built using the features of the SPEEDES framework. For instance, the low-fidelity sensor model is a derived representation of the DDM SPEEDES function. The target model is a simple simulation object that has position attributes. The communications are implicitly modeled as part of the swarm behavior: the position update function. Therefore, it can be concluded that development of supporting models is accomplished in this research.

Define specific reconnaissance scenarios to be used with this research. Three scenarios are developed and tested in the course of this effort. While not all parameters that influence a real world reconnaissance mission are part of the scenarios, a basic reconnaissance framework is used. From the results in Chapter 6 Section 6.6.1 this sub-objective is accomplished.

Define effectiveness criteria for each scenario (metrics). Three main metrics associated with the reconnaissance mission are developed and categorized under measures of effectiveness and measures of performance. These measures are used in the testing effort to evaluate the effectiveness of swarming reconnaissance. This sub-objective is accomplished in Chapter 3 Section 3.2.

Simulate each reconnaissance scenario for mission effectiveness. As mentioned above, the pedagogical, passive, and active scenarios are simulated and measured with the defined metrics. The results from Chapter 6 Section 6.6 indicate that swarming reconnaissance is a viable candidate for future military missions. Thus this sub-objective is accomplished.

7.2.3 Overall. A basic parallel swarm simulation system is now available to the research community. While it is not as efficient as one might desire, the swarming function is accurate and the design is flexible for future improvement. This research introduced the idea of using swarms for military reconnaissance. Using UAVs address the safety requirement by keeping humans away from the front lines of the battle when performing reconnaissance. It has been demonstrated that swarming reconnaissance is one candidate that addresses the future requirements for a safer, more accurate, and more fault-tolerant weapon system.

Some meta-level conclusions address the simulation system and software engineering. One could observe that the amount of inter-process communication required for a distributed simulation of many objects with frequent interactions does not perform well in a discrete event simulation. Surprisingly, this is by design of the discrete event system. Discrete event simulations are built for systems that model only discrete events (low frequency) in a system–a relatively low number when compared to the total number of events in a system. This is not an issue until the number of simulated objects is large (> 100.) Because the number of discrete events is almost the same as the total number of events for the swarm model, the simulator is overwhelmed causing poor

performance. An interesting study is to use this parallel model in a fluid dynamics simulation or particle simulation, where the number of expected interactions is designed to handle a much greater number of events than that of a discrete event simulator. The conclusion about software engineering is positive. Because the serial code used an object-oriented architecture, the porting from one system to another was relatively simple. Thus when considering parallelizing existing software models, a priority should be placed on code that has an object-oriented structure. This is consistent with software engineering principles.

7.3 Contributions

Current research has not seen a parallel implementation of a swarming system. This unique contribution is even more pivotal because the framework upon which it is built includes support for discrete event simulations. This allows for quickly integrating higher fidelity or specialized support models as needed. A second contribution is applying the idea of emergent behavior as a technique for accomplishing a reconnaissance mission. Currently, there is no evidence of that application outside of this research effort. Third, a taxonomy for reconnaissance is contributed. And fourth, a comparison of swarming simulators resulted in establishing desired criteria for a swarm model and summary of current swarm-related simulators against that criteria. One sponsor, AFRL/SNZW, has already included the Linux port of this swarm model during a simulator demonstration recently presented to Congress. Also, a contract has been awarded that includes a requirement to integrate the swarm model into the IDAL framework. Therefore, this research has provided an important step toward making swarming reconnaissance a reality for today's military.

7.4 Future Work

Swarm related characteristics (swarm size, computational ability, sensor load, dimensions) that are most effective for various reconnaissance scenarios can be characterized with the simulator. Stream-lining the parallel swarm implementation can be accomplished by using SPEEDES diagnostics tools to identify those long-pole characteristics of the simulation and then redesigning that bottleneck. Incorporating way points and threat avoidance into the swarm model can provide for higher fidelity simulations. Those added features can then be used to develop a search algorithm to direct the motion of the swarm performing reconnaissance so that zone, route, and

battle damage assessment operations can be simulated. Additional supporting models can be integrated to increase the overall fidelity of the simulation, such as a high-fidelity infrared/radar sensor, packet-level communication model, on-board sensor processing, data fusion, and adding support for modeling 3D interactions. Another future task is to research existing reconnaissance missions and recreate them, but instead of using the original technology insert the swarming reconnaissance model and compare the simulated performance to that of the original.

Appendix A. High Performance Computing Systems

Several Beowulf systems available at the Air Force Institute of Technology (AFIT) commonly used for parallel computing applications are described below. Details of the control structure, memory architecture, interconnection network, and scheduling software are presented. Table 15 lists the Beowulf systems that are being described. Four systems are shown but two of them share the same CPUs (ASPEN, MYRINET). The faster Myrinet backplane allows for even more efficiency when processing tasks in parallel.

A.1 Processing Control Structure

The Aspen Beowulf platforms are configured as multiple instruction multiple data stream (MIMD) computers [43]. This means that the processing units on these MIMD computers work independently because each node in the architecture has its own control unit. MIMD architectures are capable of executing a different program independent of the other processing elements in contrast to single instruction multiple data stream (SIMD) computers. The MIMD architecture offers a great advantage over SIMD when running a parallel discrete event simulation in which many separate concurrent processes must communicate through a high speed backplane.

A.2 Memory Architecture

The memory architecture of the Beowulf system has a large impact on how the parallel programs are coded because of the impact of read-write accesses. Aspen is equipped with both uniform memory access (UMA) and non-uniform memory access (NUMA) [43]. The UMA architecture is where each processor has equal access to any memory segment which means that the physical memory (RAM) is shared among various processors. This implies that when one processor has write access to a block of memory, then it must maintain a semaphore scheme to prevent another processing element from overwriting that same block. NUMA is the familiar architecture common in every desktop computer. Each processing element has its own private memory, hence there is no shared memory, thus writing to memory does not require the overhead of synchronizing read-write accesses. NUMA applies to the all 48 nodes of Aspen, where each node has UMA access between the on board dual-processors.

Table 15 AFIT High Performance Computing Systems

HPC System	**Operating System & Backplane**	**Processors**	**RAM (Gbytes)**	**Node Specifics**
ASPEN	Redhat Linux 7.3 Ethernet	Pentium 3, 1GHz	1	32 total nodes, 2 processors per node
MYRINET	Redhat Linux 7.3 Myrinet	Pentium 3, 1GHz	1	subset of ASPEN (16 nodes) 33-48
Pile of PCs	Redhat Linux 7.1 Ethernet	7 nodes: Pentium 4, 1.7 GHz 8 nodes: Pentium 3, 933 MHz	0.256 0.512	15 total nodes, 1 processor per node
POLY	Redhat Linux 7.1 Ethernet	AMD Athlon, 1.4 GHz	0.768	16 total nodes, 1 processor per node

A.3 Interconnection Network

The interconnection topology of the AFIT parallel system is classified as dynamic. The AFIT clusters are all based on a switching network in which communication links are connected to one another by the switches. This is representative of a crossbar network with the following characteristics:

- Diameter: The maximum delay that a message encounters when communicating between a pair of nodes: 1
- Bisection Width: The minimum number of communication links that must be removed to partition the network into two equal halves: p, (number of processing nodes).
- Arc Connectivity: Connectivity is defined as the multiplicity of paths between any two nodes. Arc connectivity is then defined as the minimum number of arcs that must be removed from the network to break it into two disconnected networks: 1
- Cost : Number of required communication links required by the network: p^2
- Routing : Cut-through

For the above reasons, the switching interconnect at AFIT is considered one of the faster interconnects available. For more network interconnect details consult [43].

A.4 Scheduling Software

The software installed on Aspen is PBS Pro 5.1, which is a scheduling software package for Linux parallel computing clusters. PBS stands for "Portable Batch System." It allows many users to share a cluster of machines by dynamically queuing job requests and assigning nodes to those queued jobs. Interfacing to this software had a significant design impact on the scripts that execute the experiments. See the user guide [52] for more information.

Appendix B. Selecting a Simulator for Modeling UAV Swarms

B.1 Introduction

Simulating a swarm of unmanned aerial vehicles (UAVs) might at first appear daunting. However, this is not the case when considering all of the source code that is freely available. The first task in simulating a UAV swarm is selecting the simulator that upon which the UAV swarm model is to be executed. This paper reviews the current simulation programs that can potentially be used to model swarms of UAVs. A brief introduction to the problem domain is given first. A set of desired simulator characteristics including a fidelity evaluation is defined. The review then considers 16 potential candidate simulators. Upon evaluation of each of the simulators against the desired characteristics, a selection is made.

B.2 Problem Domain

B.2.1 Swarms of UAVs. The use of UAVs in a future military environment includes the using miniaturized, sensor bearing, processor intensive, network of micro-UAVs for a high-threat environment. The value of such swarms is that each unit is dispensable without threat to human life and has low cost. The Navy has already spent millions of dollars to understand how this micro-UAV could be a reality [93]. Its mission is that of reconnaissance to detect hostile forces and materials.

Swarm or emergent behavior systems present a unique implementation method for a sensor system with a large number of individual sensors. Swarm behavior, like that seen in bee swarms or flocks of birds provides a stable organization of sensor platforms that is flexible, able to adjust rapidly to changing environmental conditions. These UAV wireless mobile node systems also provide graceful degradation when individual sensors fail. The communications system for the sensor network must provide for the effective and efficient transfer of large amounts of data in a highly dynamic network environment.

Many more details on the specific subject of swarming of UAVs can be found in [26].

B.2.2 Discrete Event Simulation. In addition to the problem domain of this paper's subject there also exists the problem domain in general for simulations. A simulation is "the

Table 16 Simulation Components of a System

Term	Definition
entity	object of interest in the system; it requires explicit representation in the model
attribute	property of an entity
activity	a time period of specified length with a known start time
sytem state	the collection of variables necessary to describe the system at any time, relative to the objectives of the study
event	instantaneous occurrence that may change the state of the system
event notice	record of an event to occur at the current or some future time; includes any associated data necessary to execute the event
event list	list of event notices for future events ordered by time of occurrence
endogenous	activities and events occurring within a system
exogenous	activities and events in the environment that affect the system
system	collection of entities that interact together over time to accomplish one or more goals
model	abstract representation of a system; containing stuctural, logical, or mathematical relationships which describe the system in terms of state, entities, sets, processes, events, activities, and delays
list (queue or chain)	collection of associated entities, ordered in some logical fashion
delay	duration of time of unspeci ed inde nite length, which is not known until it ends
clock	variable representing simulated time

imitation of the operation of a real-world process or system over time" [12]. It involves exploring the behavior of that system by developing a simulation model. Discrete event simulation simply means that only particular events of interest are included in the simulation rather than mimicking every single event that occurs in the corresponding real-world process. Said another way, it is the modeling of systems in which the state variable changes only at a discrete set of points in time [12].

Simulation requires that the system be decomposable into objects that make up the simulation. Each of the objects must include the attributes necessary to imitate what occurs in the real-world system. Common to all simulations are several terms that aid in this system decomposition. Table 16 presents the terms along with their definitions as discussed in [12].

It should also be noted that the simulation terms presented in this section are exclusively for dynamic and stochastic systems, meaning that time causes the state changes and the system contains random elements.

Table 17 Summary of Simulation Characteristics

category	*characteristic*	*desired value*
simulator	operating system	linux
	open source	yes
	documented	yes
	source code	C++
	software design	object-oriented
	graphical interface	yes
	command line mode	yes
	built-in statistical analysis	yes
	data monitoring/custom analysis	yes
	built-in visualization tools	yes
models	environment (3D, 2D, or 1D)	high fidelity
	target	high fidelity
	vehicle (UAV) communications (wireless ad-hoc)	high fidelity high fidelity
	sensor	high fidelity
	data fusion	high fidelity
	vehicle control (swarm behavior)	high fidelity
high performance	allocate tasks to multiple processors	yes
	supports message passing between simulators	yes
	parallel platform	yes
	scalable up to thousands of entities	yes
	state-saving support	yes

B.3 Simulation Characteristics

Several characteristics outlined below describe the ideal UAV swarm simulator and is used as a measuring stick for the simulators evaluated in this paper. Table 17 presents a summary of what is discussed.

B.3.1 Program Characteristics. Ideally the simulator is a Linux-compatible distribution that is open-source with comprehensive documentation and has a large user base. It is implemented in C++ and allows for easy modifications–object oriented design. Reuse constructs such as inheritance, extension, and structures are essential. The only limitations on capabilities should be related to the hardware on which it is run.

It has both a graphical user interface for ease of learning the tool as well as command line capability for scripting simulations and parameter specifications. It includes built in statistical

methods that can automatically analyze simulation results, in addition, other metrics development can utilize the built in analysis tools. Probing mechanisms that allow for monitoring of data without interfering with simulator performance is expected.

A visualization suite of tools that augment the simulation should be available. This tool set should allow for visualizing the simulation in a time-stepped or event-based fashion. Customizable views/panes should allow for each individual viewer to be tailored for the particular interest area. Multiple data views should not affect performance.

The authors of [102] provide a taxonomy for designing computer-based simulations. They discuss several levels of the taxonomy of simulation including parallel and distributed systems, usage, simulation, simulation engine, modeling framework, programming framework, design environment, user interface, and system support. Many of these taxonomies are contained within a subset of the characteristics just discussed, however, the usage taxonomy brings into light the important aspect of fidelity.

B.3.2 Simulator Fidelity. All simulators are not created equal. Many have greater levels of detail for using in particular application areas that others assume away. Some simulations mimic the real-world system at a low level for the purpose of obtaining an initial low-fidelity evaluation, while high-fidelity simulations are used to model high-risk systems for safety reasons or when the system is simply too expensive or complex to research otherwise. Achieving the highest level of fidelity possible is the difference between simulators and emulators. An emulator executes exactly like the physical world system itself–no assumptions, no unknowns. Typically, deterministic systems can be emulated, for example operating systems are purely based on deterministic logic and thus can be emulated.

Although the modeling and simulation community commonly use the word fidelity, there is no widely-accepted definition or method for measuring fidelity [66, 88, 90]. The reason for such disagreement is due to the subjectivity of comparing a simulation model with a real world system. Often fidelity is quantified based on the intended application or *focus* of the simulation model. Thus a flight simulator focused on training pilots that is given high fidelity is considered to be a low fidelity simulation model that is focused on the application of an aircraft maintenance logistics simulation. Thus fidelity depends on the application focus. For the purposes of this paper the

focus is the efficiency and effectiveness of performing a swarm reconnaissance mission. In [66], the fidelity of the simulation model is linked to the exercise's measures of performance (MOP) and effectiveness (MOE.) Each aspect of the model is graded based on the impact to measures of performance. For example, in a communications environment a typical measure of performance is bandwidth. The fidelity of the simulation then depends on whether or not the modeled bandwidth is impacted by environmental affects as would occur in the real-world. However, if the focus of the network simulation is for an area that is not susceptible to adverse environmental affects, then that aspect of the model would not be necessary to meet a higher level of fidelity.

The fidelity of the simulation model determines the accuracy in which the real-world system is modeled. For the purpose of comparison in this paper, each model is rated either high, medium, or low. A high rating indicates the resolution of the model accurately represents every significant aspect the real-world system, low indicates many approximations or assumptions are included and as such a Monte-Carlo approach has been taken, and medium mixes both. While [88] states that "the comparison of simulation results with real-world data is conceptually the most robust approach for fidelity," that comparison only applies when the corresponding real-world data is available. In this case, the design is modeling a system that does not yet exist, thus fidelity must be measured another way.

Table 18 contrasts high, medium, and low fidelity for a Swarm Reconnaissance focused application. Each component of the swarm application is presented with corresponding levels of resolution that a simulation model might include. For example, a high fidelity vehicle model includes not only such basic aspects as orientation and velocity but also the physics behind the 3-D simulation and drift correction algorithms for environmental influences like high-crosswinds. Similarly, a low fidelity swarm behavior model might only provide random behavior using global state variable with a fixed size neighborhood. The medium fidelity swarm behavior model would extend the low fidelity model by providing operation modes in which a desired behavior can be achieved as well as stability algorithms that account for situations where one or more members are removed from the swarm perhaps by a target with attack munitions. Quantitative fidelity measures are presented also for the target, communications, and sensor models that are all part of a Swarm Reconnaissance focused application.

Table 18 Swarm Reconnaissance Fidelity Model Characteristics

Application	Resolution	Fidelity			Application	Resolution	Fidelity		
vehicle					*communications*				
	orientation	L	M	H		traffic flow	L	M	H
	velocity	L	M	H		protocol type	L	M	H
	position	L	M	H		routers	L	M	H
	sensor tasking	-	M	H		packets	-	M	H
	damage level	-	M	H		equipment type	-	M	H
	physics-based	-	-	H		physics-based	-	-	H
	drift-high crosswind	-	-	H		error correction	-	-	H
swarm behavior						redundant	-	-	H
	random	L	M	H		messages	-	-	H
	global state	L	M	H		electronic interference	-	-	H
	size of neighborhood	L	M	H	*sensor*				
	stability	-	M	H		sensor type	L	M	H
	operation modes	-	M	H		range	L	M	H
	decentralized	-	-	H		damage level	L	M	H
target						electronic interference	-	M	H
	dynamic	L	M	H		orientation	-	M	H
	target type	L	M	H		physics-based	-	-	H
	jamming modes	L	M	H		sensor fusion	-	-	H
	coordinated	-	M	H					
	physics based	-	M	H					
	attack munitions	-	-	H					

B.3.3 Modeling Characteristics. When considering a model for a UAV swarm several models are needed to produce a complete picture. Those models include one for an environment, target, ground control center, vehicle, a communication network, sensor fusion algorithm, on-board sensors, and swarm behavior.

- Model Descriptions

    1. To imitate a real-world swarm of UAVs, an environment model is crucial because of the external influences and 3-dimensions that reality dictates. Therefore the environment model is one that acts as a container for the swarm of UAVs as well as all other objects in the real-world such as hostile targets and significant terrain features such as large bodies of water and mountains. It provides the frame of reference for all positioning information such as longitude and latitude measurements for vehicles, targets, and other objects of interest.
    2. A target that is present in the environment has a specific location, purpose, and health monitor. Thus a target model must be able to hold a position, pose a threat, interact with the swarm of UAVs, and accept damage from another vehicle. The target should also have its own characteristics of interest such as expected lifetime and autonomous activities.
    3. Similarly, a vehicle model also has its own characteristics of position, size, energy management, control, on-board sensors, processing capacity, and a communications mechanism. It is clear now how that simulation decomposes a real-world system into a simple matrix of objects that interact with each other. The complexity of the vehicle model depends on how many of its characteristics are broken down into sub-models such as on-board sensors or processing capability.
    4. One of the more important models is the communications model that represents the vehicles ability to communicate. The data that passed from one vehicle to the next includes sensor data from the local area activity, positional data of the other vehicles in the near vicinity, command messages from the ground control center, as well as capability for other types of information to be passed. This communications network model is a wireless ad-hoc network and should be modeled as such. This model is

important because the efficiency/effectiveness of the communications can negatively impact the ability of the swarm to perform its mission.

5. Utilizing the communications model is the sensor fusion model. Sensor fusion is the activity of combining sensor information from multiple data sources to present a more accurate picture than any one data source. The sensor fusion model efficiently aggregates data based on a mission objective or command request. It filters out irrelevant data and relays information through the communications network to the ground control center.

6. Flying a swarm of unmanned vehicles requires an intense awareness of the nearby vehicles. As such a behavior model that prevents collisions and maintains cohesion is required. This model accounts for the present location and future movement of the vehicle in 3-dimensions. Integrated are physical limitations of the vehicle such as fuel, g-limits, and temperature. This is the vehicle behavior model.

Ideally all of these models are already integrated into a single simulation platform. They already have the ability to interact with each other and can be easily customized. Additionally, each model should have several built in predefined variants to allow for model performance comparison during simulation. For example, the network model should not only support a single wireless ad-hoc network protocol such as Directed Diffusion[49] but also standard wired protocols and other wireless variants; the environment/target models should already include several ground based and waterborn target models and have capability to support simulation in 3-dimensions *or fewer* if desired; the behavior model should also have various predefined standard configurations to choose from including swarm variants such as the one discussed in [53].

B.3.4 High Performance Characteristics. Sizes of swarms range from tens of UAVs to thousands or more depending on the mission. The amount of data collected by each of these UAV's sensors can overwhelm current real-time processing systems already. Modeling this complex system with the details of packet-level communication, data fusion algorithms, power consumption calculations/estimations, and 3-dimensional positions can quickly use up all available processing resources. Thus to mitigate the large computational load, high performance systems are the computing resource behind the simulation.

The final characteristics are that of this high performance simulation platform. This simulator allows for allocating tasks from a simulation to multiple processors simultaneously. Message passing between concurrent simulations must be available. Scaling the simulation from tens to thousands should result in no adverse affects in effectiveness or efficiency. The simulator has built in state-saving manager to avoid loss when a hardware failure prematurely ends a simulation. Fundamentally what has been described is a parallel discrete event simulator.

B.4 Simulators

The list of available simulators is endless, however to find one that is useful for the purpose at hand is the task of this paper. An overview of several relevant simulators is presented briefly first, then a comparative analysis is made based on the previously stated criteria. Finally, a selection is made.

B.4.1 Overview. What is hoped to be captured in this overview are the key characteristics of the software package as well as enough background information for comparison with other simulators. This section is categorized into 3 areas. The area corresponds to the simulator's target application. Those three applications are swarms, networks, and simulation frameworks. Table 19 presents a quick overview of all discussed simulators.

Table 19: Overview of Relevant Simulators

tool	Description	Developer	Application
swarm	Kadrovach's swarm simulator generates particle position information based on user specified parameter inputs and displays a real-time swarm animation. Also developed is the automatic translation of the movement data output file so that it is compatible with ns2 enabling evaluation of wireless communication protocols within a swarm.	Air Force Institute of Technology	Swarm communications
swarm (lua)	Chin Lua's Simulator demonstrates a reactive, synchronized, swarm of UAVs that executes a multi-point attack against a target.	North Dakota State University	Swarm simulators
Simulation	ICO Systems very configurable simulator is an agent-based swarm model of UAVs flying over a search area populated with targets (moving or stationary.) One of the many simulated on-board sensors is a phermone detector being used (as an ant would) for decentralized control.	Air Force Research Laboratory, Control Sciences Division	Swarm simulators

Table 19: *continued...*

tool	Description	Developer	Application
TextSwarm	James Lotspeich's TextSwarm simulator generates swarm member position information and metrics resulting from a swarm model that is built on behavior matrices and rule sets. The swarm follows a pre-planned path through a territory filled with goals and threats.	Air Force Institute of Technology	Swarm simulators
MultiUAV	Developed by Veridian, this 3D Swarm Simulator requires the use of Matlab and Simulink tools. It simulates UAVs and targets with extensive embedded flight software management.	Air Force Research Laboratory, Control Theory Optimization Branch	Swarm simulators
SWEEP	This simulation platform uses templates to control the characteristics of swarm behavior, environment, and interactions between them. The application was developed for understanding how UAVs are used in chemical cloud detection.	John Carroll University	Swarm simulators
ANSim	Hellbruck's program is a graphical simulator based on a simple transmission model for statistical simulations of Ad-Hoc Networks. It includes outputs that will interface to ns2 and glomosim.	International University, Germany	Network Simulators
Cougar	An approach to tasking sensor database networks using in-network computation. This simulator can interface to live sensor hardware. This is DARPA research project.	Cornell University	Network Simulators
H-MAS	Based on the swarm toolkit, this simulator provides a workspace to evaluate a variety of mobile ad-hoc network simulations at the physical, medium access, network, and applications layers.	University of Notre Dam	Network Simulators
ns2	This detailed network protocol simulator is targeted at researchers. The open source distribution includes several validated existing network protocols and allows the user to quickly change configurations and simulate a number of scenarios.	University of California, Berkeley	Network Simulators
OMNeT++	OMNeT++ has tools to support many phases of a network simulation project, from graphically designing the topology to writing the models components, from debugging and verification to full-speed execution and visualizing the results.	Technical University of Budapest	Network Simulators
PDNS	A parallel version of the network simulator, ns2 allows for a distributed simulation enhancing simulaton performance and scalability.	Georgia Institute of Technology	Network Simulators

Table 19: *continued...*

tool	Description	Developer	Application
SWAN	The Simulation of Wireless Ad-hoc Networks (SWAN) simulator is designed to simulate detecting the status chemical, radioactive, or other catastrophic agents within a geographical region.	Dartmouth College	Network Simulators
GloMoSIM	A simulation library written in PARSEC for high-fidelity simulation of large-scale wireless network models. PARSEC is the C-based simulation language for parallel execution of discrete-event simulations.	University of California, Los Angeles	Frameworks
HLA	An architecture specification written for the DoD to support interoperability and reuse for different types of programs, specifically simulations. This structure depends upon a runtime infrastructure to synchronize heterogeneous simulations.	Department of Defense, USA	Frameworks
SPEEDES	SPEEDES-based object-oriented simulations that run on High Performance Computing (HPC) platforms are able to address extremely complex problems and still maintain short run times. This mature parallel discrete event simulator is performance tuned and has built-in HLA support.	California Institute of Technology	Frameworks

B.4.1.1 Swarm Simulators. These simulators are designed expressly for the purpose of modeling one or more aspects of swarm behaviors. All of them assume the members of the swarm to be UAVs.

B. Anthony Kadrovach's Swarm Simulator: swarm. Kadrovach [53] developed his simulator to characterize the behavior of swarms. This graphical program simulates along with a real-time animation any number of particles (based on system memory–30 starts to slow down the animation on a 2GHz, 1G RAM Pentium machine) based on several input parameters such as boundary repulsion weight, peripheral weight, comfort zone, alignment weight, attraction weight, neighborhood size, velocity factor, maximum turn radius, and maximum turn perturbation. This program was designed and used by Kadrovach to generate swarm data to develop a swarm behavior identification metric that provides for a quantitative methodology for global swarm behavior characterization. He incorporated the resulting swarm movement data into the *ns2* network simu-

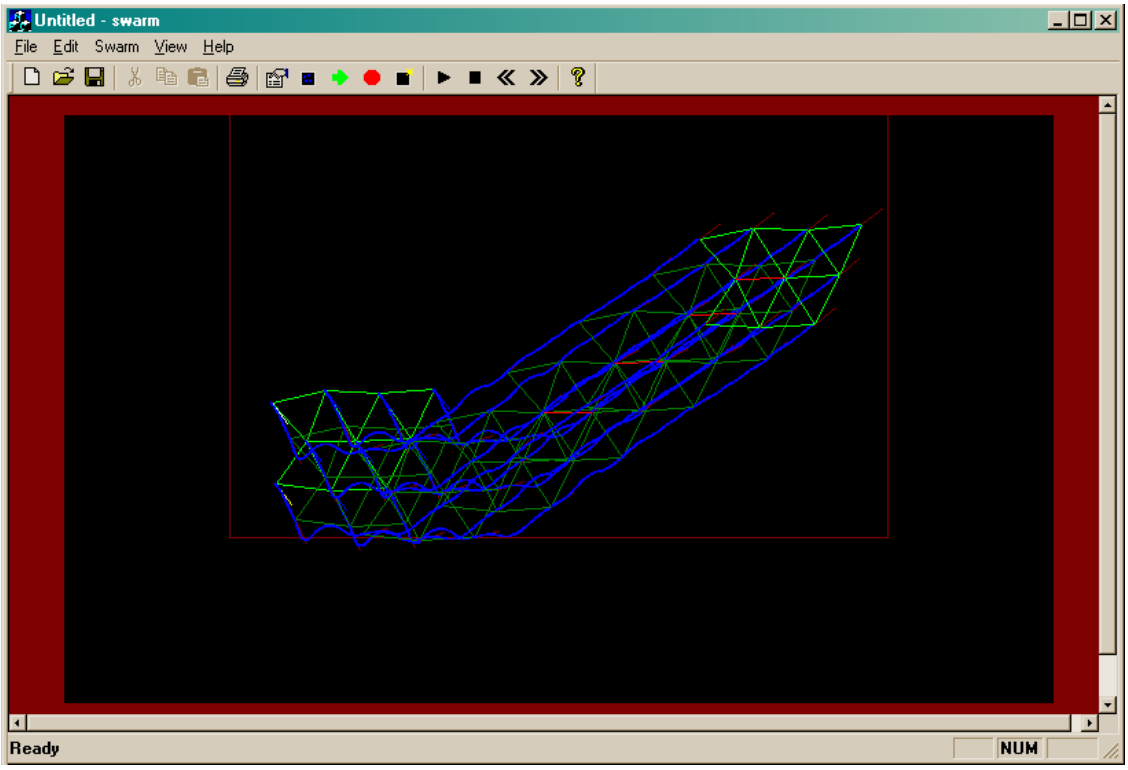


Figure 25 Kadrovach's Swarm Simulator GUI

lator to evaluate the network communication dynamics for the directed diffusion protocol among others.

Also incorporated in the swarm model is a vision blocking perspective for each participating member. This means that the influence that each swarm member has on every other member of the swarm is limited to who can be seen, which is similar to how birds fly in a formation. Only the birds that are in front can influence the current bird's decision to move in a particular direction. Similarly, Kadrovach incorporates this vision model for the particles in his swarm program.

Figure 25 shows the graphical user interface (GUI) for this swarm package. The majority of what is seen in this animation is not simply the movement history of the members of the swarm but the distance intervals between each particle based on the comfort zone. As shown in Figure 25, the green joining lines indicate the distance between the two particles is within the comfort zone; the red indicates the particles are closer than desired, and the absence of a joining line indicates the particles are out of communication range (this was designed for integration into the

ns2 communication model.) The animation is not the only data generated with the program. Each particle's position information is recorded externally and made available to the user. For the serious user, a command line interface is available that produces the position information without the overhead of the processing cost of animation.

Chin Lua's Swarm Simulator: swarm applet. Lua's [63] simulator is the first found that simulates a swarm that attacks a target. The focus of the simulation is wholly upon autonomous search and attack. He develops a control strategy that surrounds the stationary target and then delivers the munitions from the UAVs. This is all done based on passive short range sensors and simple, inter-agent communication. This is a work currently in progress at the North Dakota State University in affiliation with the Navy. The only working version of the program available is a Java applet. The author declined a request for the source code. Demonstrations are available at his website [63]. A sample 18-point attack is shown in Figure 26.The outer circle is the line upon which the UAVs surround the target (red dot.) After the swarm locates the target then aligns themselves around the outer circle then the attack begins.

ICO Systems Swarm Simulator: Simulation. Developed under a small business innovative research contract for the Air Force Research Labs, Icosystems authored this simulator [41]. This UAV swarm simulator uses an agent-based model flying over a search area that has targets. It uses decentralized control strategies to include simulated pheromone trails which are mimicked after ant colonies which use real pheromones to collective determine the shortest path to a resource. Included is both 2D and 3D visualizations with the option of running in real-time. Each UAV is equipped with various sensors and can fly at variable speed with independent pitch and yaw control. Communications can be simulated using either global or local strategies.

This program can be invoked with a GUI as shown in Figure 27 or run from a command line. It is configurable through a mixture of user specified parameters on the command line or through the GUI widgets. The simulator is written in Java.

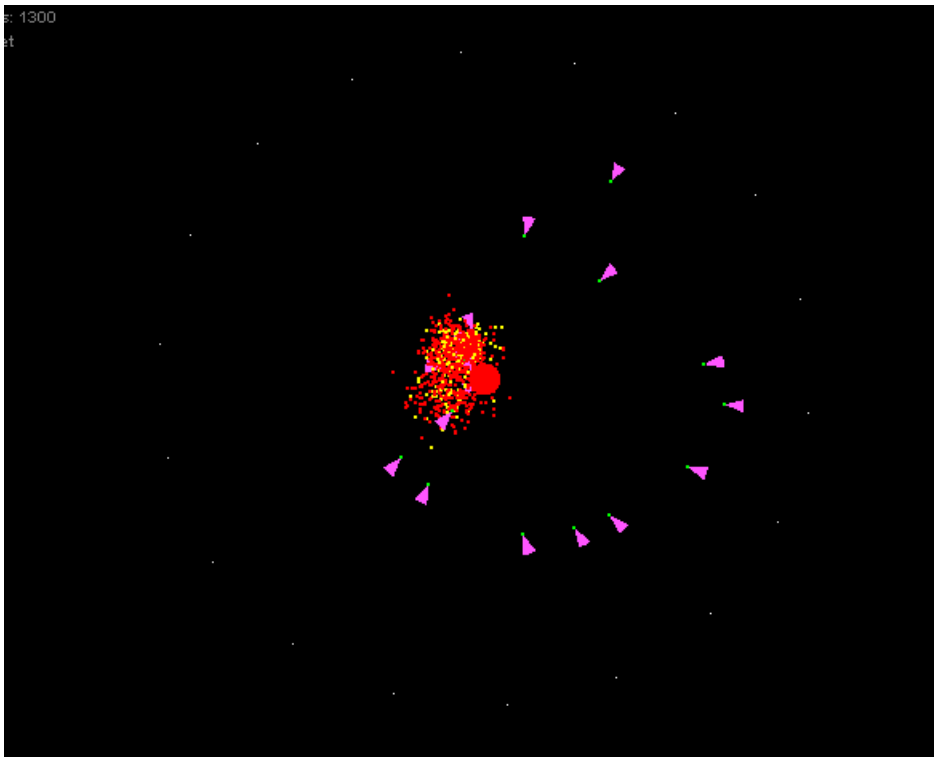


Figure 26 Lua's Swarm Attack Simulation

Figure 27 is a screen shot of the UAV simulator with 200 members in the swarm with 5 targets in the environment. It is described below.

> The simulator is divided into three areas. The top-left area includes various widgets to control certain aspects of the simulation in real time, such as pausing and restarting, shuffling targets randomly, or modifying the dynamics of the UAVs. This area also displays the time elapsed, percentage of terrain covered, and percentage of targets identified.
>
> The bottom area is a 3D view showing the boundaries of the terrain being searched (black wire-frame box), the UAVs (blue circles), the area swept by the UAV ground/target sensors (yellow triangles), and the targets (red/green squares). Each UAV has a red vertical line connecting it to the ground to help visualize its position, a black line indicating its current heading, and a yellow line indicating its desired heading.
>
> The top-right area is a top-down matrix representation of the terrain, which shows the grid used to determine coverage, the x,y position of the UAVs (black) and targets (red if not found, green if found), and a blue trace of varying intensity that represents the pheromone, i.e., the degree to which a given cell has been flown over by UAVs [41].

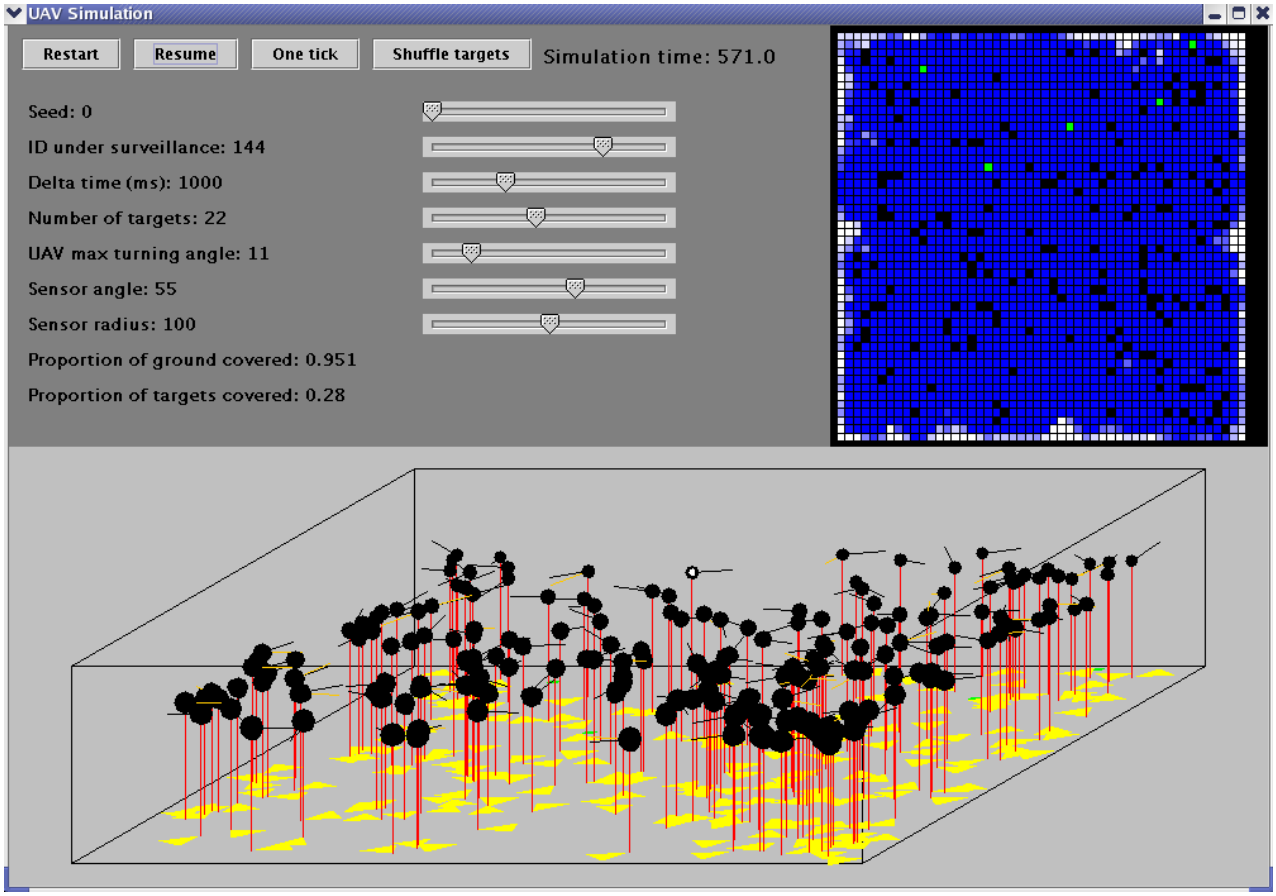


Figure 27 Ico Systems GUI Interface

James Lotspeich's Swarm Simulator: TextSwarm. Lotspeich [62] developed swarm simulator for the purpose of controlling swarm behavior. He uses a swarm model that accounts for cohesion, separation, threat avoidance, and goal seeking each with its own weighting factor. By varying the weightings of each of the components that contribute to the direction vector a corresponding swarm behavior is realized through potential fields. His simulator is coded in Java and does not include a GUI–with the exception of a separate visualization tool that can be used post-simulation.

The program uses a landscape file along with a behavior matrix as inputs to the swarm simulation. The outputs include positional data for each swarm member at every simulated time step along with a metrics file that is tailored to Lotspeich's thesis research. Way-points control the path of the swarm members, threats can be part of the landscape, and one or more goals are also part of the landscape. Built-in to the program are the working behavior matrices and landscape files–all of which can be customized. There are no required command line parameters to change the swarming behavior–that is the reason for the behavior matrix. A separate program which implemented evolutionary algorithms was used by Lotspeich to create the behavior matrices.

Figure 28 shows a sample visualization of the output of TextSwarm. In it you see the Java interface to the visualization tool with the standard video playback tools across the bottom along with some extra visual display controls along the right (not all check boxes are functional in as of 13 Oct 03). The main window shows a center diagonal line with periodic X's which represent the way-points that the 5 member swarm is following. The red circle represents a threat that needs avoiding and the blue circle the goal. The programmed behavior is en-route–meaning avoid all threats and get to the goal. Two other behavior modes are available: reconnaissance, which attempts to perform a low-fidelity scan of a large area, and scan, which is similar to the synthetic aperture radar strip mode of the Air Force's Global Hawk radar. Lotspeich investigated the possibility of scaling his simulator up to 1024 members of the swarm but did not comment on the affect this scaling had on the performance of the simulator or visualization tool.

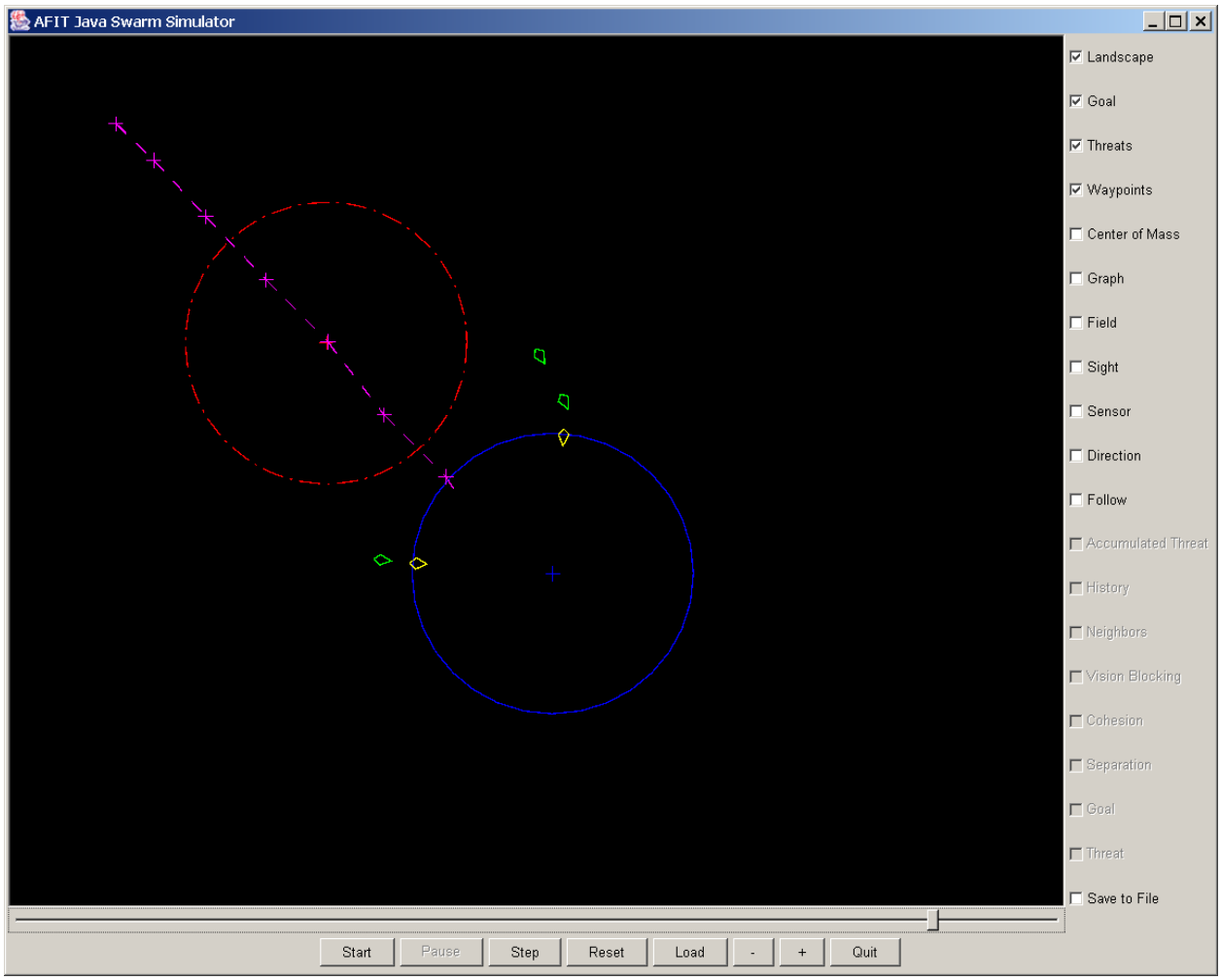


Figure 28 A Sample TextSwarm Visualization

Veridian's 3D Swarm Simulator: MultiUAV. The Veridian [8] simulator uses UAVs (up to 8 efficiently) and targets to analyze swarm possibilities. The UAVs are autonomous and have global communication. The two main models implemented are the vehicle and target models. The vehicle model includes a full six degrees of freedom (6-DOF) control as well as a sensor model with automatic target recognition. Sensor fusion is performed through statistical averaging. The targets are stationary and can exist in several modes depending on whether the UAVs have engaged them.

The input parameters include number of targets/vehicles. Several managers exist in a module called the embedded flight software. These managers are responsible for tactical maneuvering, sensor performance, routing, target, cooperation, and weapons. The outputs include both target and vehicle status and positioning information. This UAV simulator is implemented with Matlab, Simulink, and C++ source code files. This means that the computer on which the program is run must have licenses for these commercial applications. It also means both command line and GUI execution is possible.

A GUI front end has been developed for quick access to common functions in MultiUAV. Additionally, a visualization is available through the Matlab GUI programming environment. Figure 29 shows the simple GUI to this program. It is the series of command buttons on the left side of the image. The background of the figure is the Simulink design area–showing the two main entities: vehicles and targets.

Using Matlab allows for easy visualization of output information as is shown in Figure 30. This part of the simulation package displays a graphical replay of the simulation information using several shapes, lines, and colors to convey information. The figure comes from the user manual [8]. Ongoing work is currently being done to create an HLA interface to this simulation thus expanding the possibilities for scalability.

SWarm Experimentation and Evaluation Platform: SWEEP. This simulation environment [80] was developed to examine swarm algorithms with a wide variety of multi-agent scenarios. The context of this platform is in an application area of chemical cloud detection. The

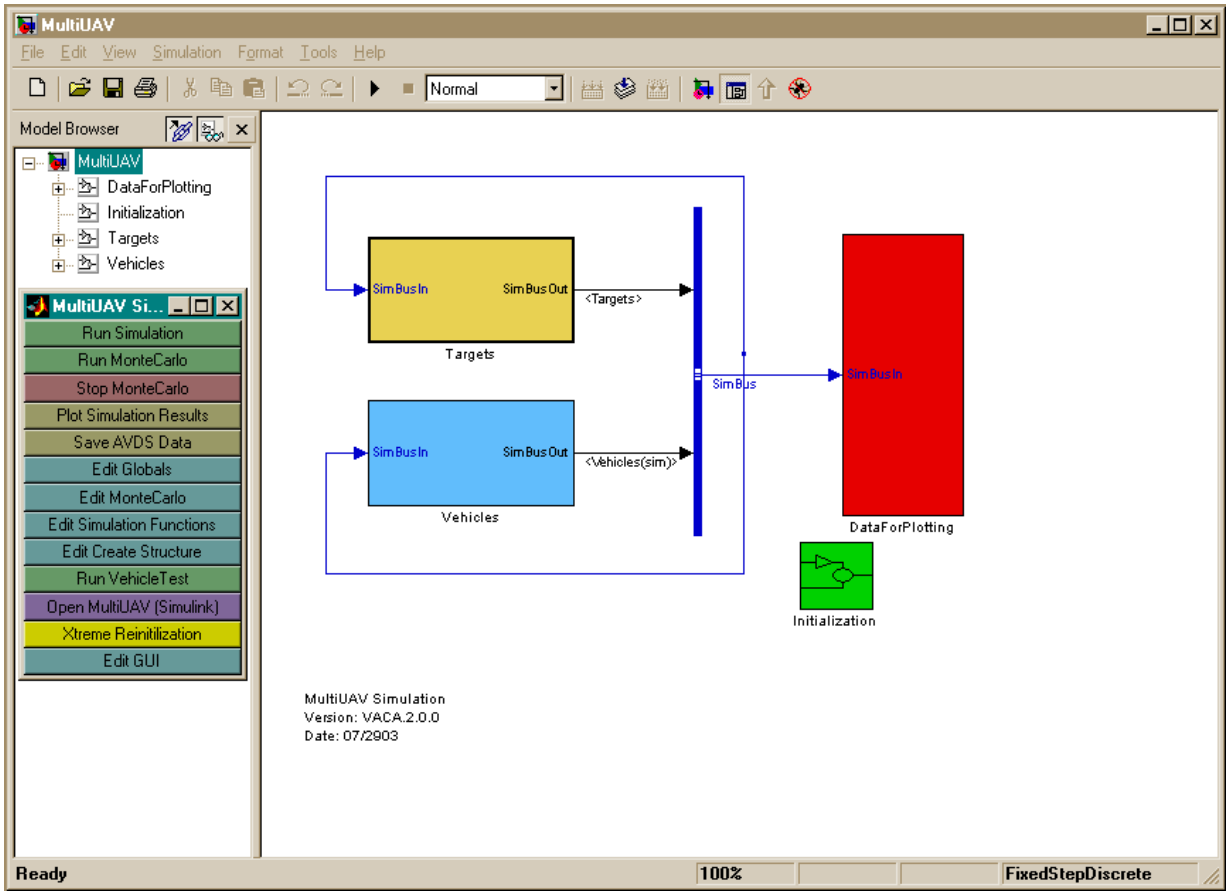


Figure 29 MultiUAV GUI with Simulink Design

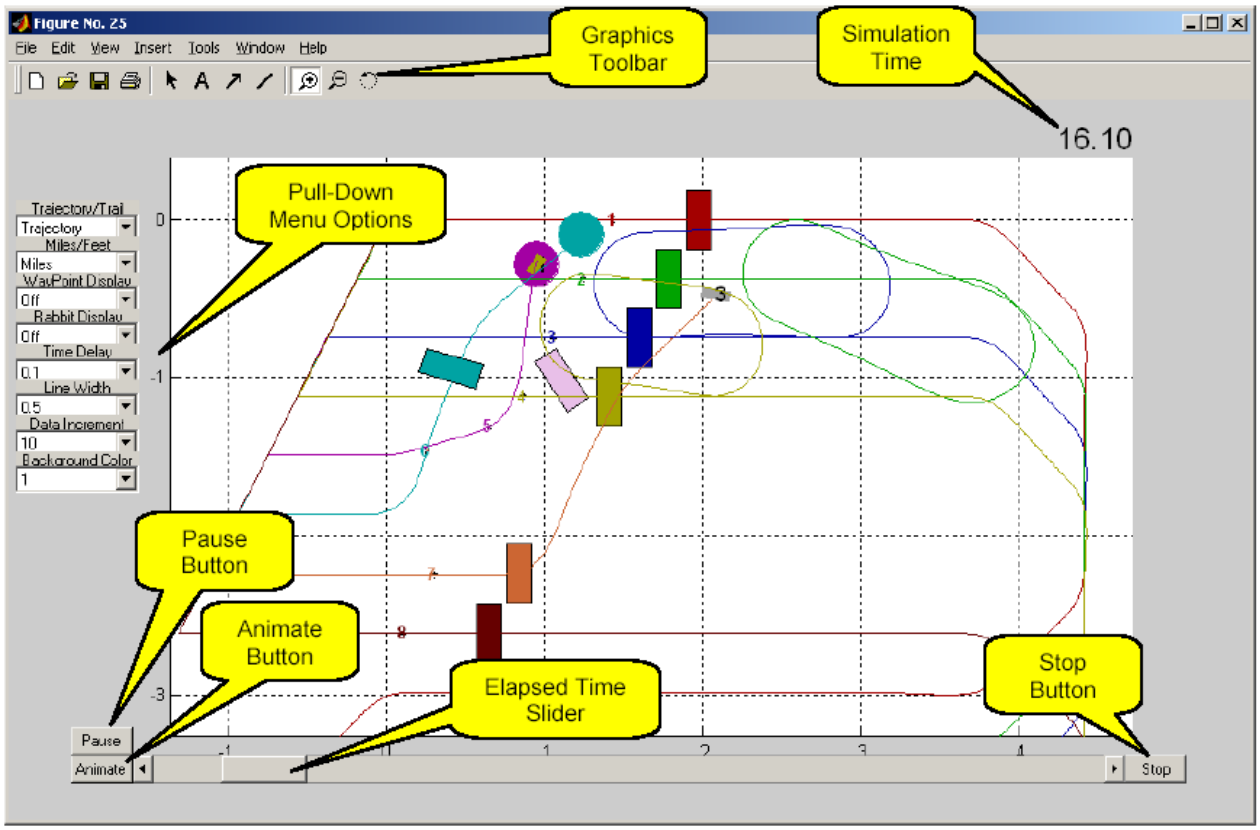


Figure 30 MultiUAV Visualization

platform consists of three basic components: a swarm of autonomous, mobile agents, a multi-faceted dynamic environment and programmable data probes. The swarm behavior is rule-based and encoded into a probabilistic finite state machine. The environment is grid-based and contains a user-specified number of characteristics which agents detect and alter (release chemicals, grabbing and depositing objects, and emitting sounds.) The agents sensors are implemented through a filter that presents local information to the agent type. The environmental conditions change over time. The data probes are user-specified experiment characteristics. The simulation records this information for post-processing.

There is mention of animation playback and thus a visualization is also part of this system. The authors mention that the execution of this SWEEP program will eventually be run in parallel as it is currently a sequential system. Running an experiment involves first "programming" several template files. The parameters in these templates control swarm behavior, environment's salient characteristics, update methods, sensor filters, and initial configurations.

The actual program executable has not been located as of the writing of this paper. Thus development characteristics are unknown for this simulator.

B.4.1.2 Network Simulators. Why network simulators? Because a swarm reduced to its purest form is simply a network. Thus a network simulation allows for swarming behavior models to be validated.

Ad-hoc Network Simulation Tool: ANSim. Horst Hellbrück from the International University in Germany produced this ad-hoc network simulation [47] as a tool for statistical simulation for practice-oriented Ad-Hoc scenarios. The user can determine the boundary conditions of the simulation by the input of some base parameters such as size and shape of the geographical area, range of the stations, etc. and receives as result e.g. the probability that two randomly selected stations are connected.

ANSim has a GUI front end that is shown in Figure 31. The program is written in Java and the source code will be available soon after Hellbrück finishes his research. The program can also

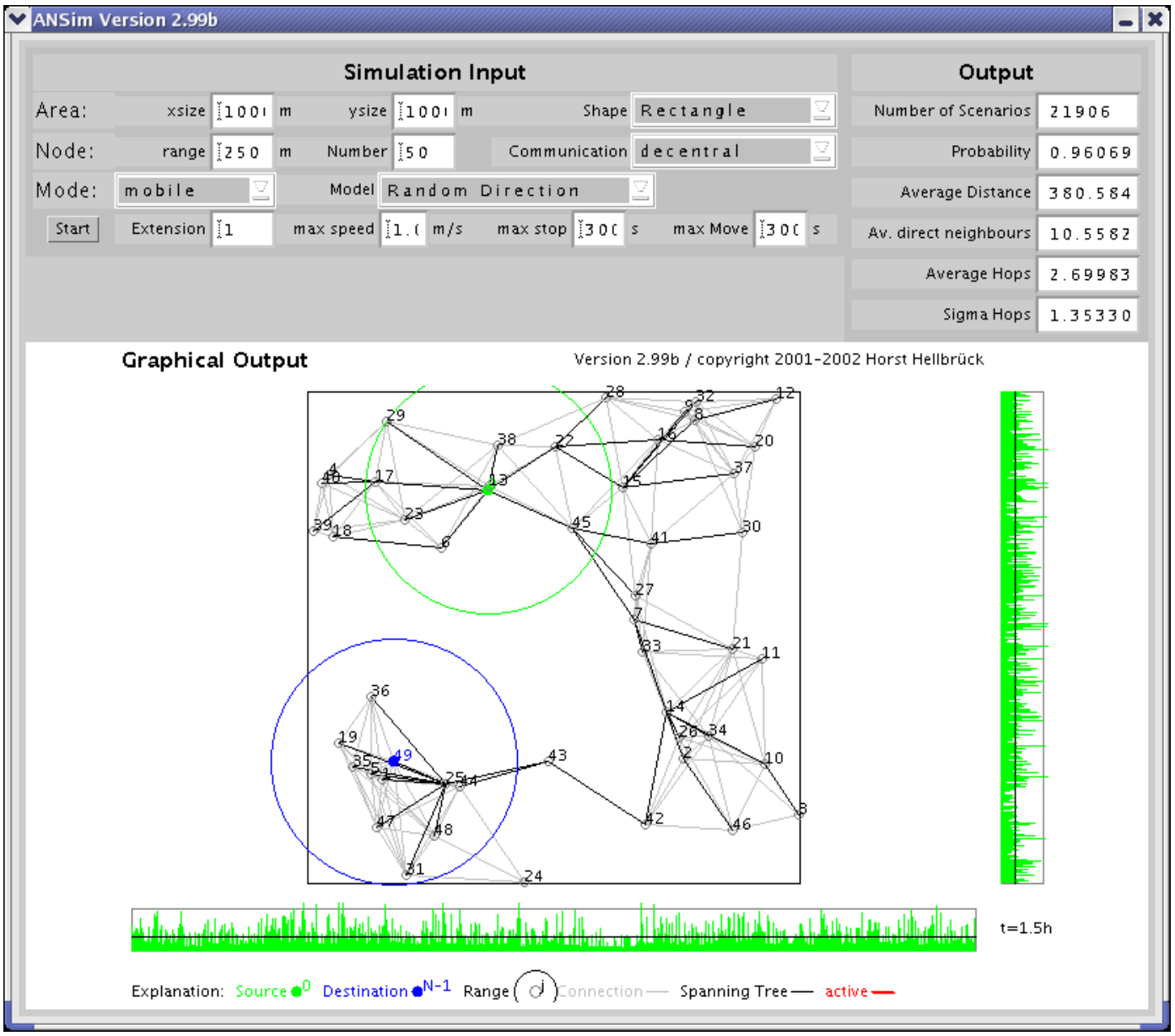


Figure 31 ANSim GUI

be executed from a command line interface with the following input parameters: area size, surface size, range of a station, number of nodes, extension of the surface on a multiple of area, communication mode, and operation mode. Additional input parameters can be specified for the static and mobile modes. The outputs include total probability that two nodes are connected, average distance of the two examined nodes, average number of direct neighbors, arithmetic average of hops to reach the target and variance, and the probabilities that the connection needs exactly n-1 hops.

The model is basically a rectangular area with N nodes. Each node has an associated transmission radius. This simple model does not account for side effects such as shading (by obstacles), reflection (at big surfaces), dispersion (at small surfaces) and diffraction (at sharp edges); only the free space loss is considered. The nodes are distributed randomly.

Cornell University Project: Cougar. The cougar project [110] is an architecture for a sensor database system. It is a loosely coupled distributed architecture that supports aggregation and other kinds of in-network computation. The simulated target network is a wireless ad-hoc sensor network for testing query processing techniques using a distributed database. Not only is this project a simulator of sorts, but specific sensor hardware (Sensoria WINSNG 2.0 nodes) has already been incorporated such that this project is a step closer to the real world than other simulators.

The system is divided into 3 parts: QueryProxy, FrontEnd, and a GUI. QueryProxy runs on each sensor node and provides the leaders among clusters of nodes, query management, and device management. Communications within the sensor network are transmitted using Directed Diffusion. FrontEnd manages the queries between the GUI and QueryProxy software. The GUI allows the user to pose queries, visually or using structured query language (SQL), and see query replies. An example SimTracker GUI from the Cornell web shows an application of this system in Figure 32. This picture shows a map with simulated targets and sensors. The sensors are blue circles and the 2 yellow and 1 red paths are the targets. An example target is a biker traveling from

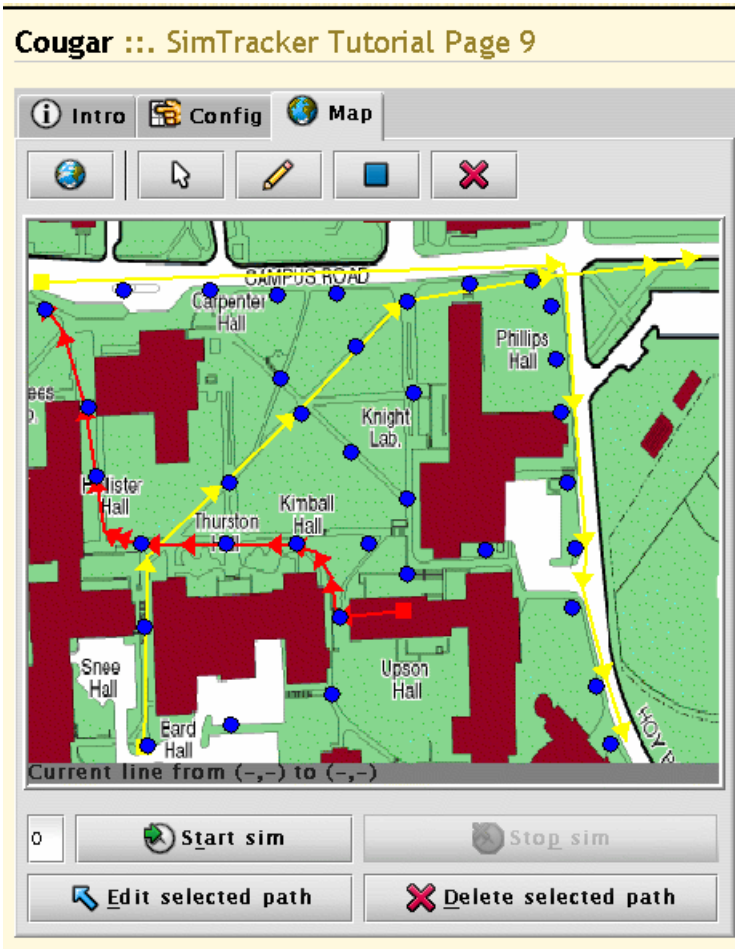


Figure 32 Cougar GUI

Bard Hall to the NNW corner of the map. The simulation provides information on which target is in range of each sensor for each time step.

QueryProxy and FrontEnd are built using C++ and the GUI using Java. The QueryProxy and FrontEnd can run under Linux or Sensoria WINS NG node hardware. Several tools can be downloaded from the Cornell website however, the source code for these programs does not appear to be available.

A Heterogeneous, Mobile, Ad-hoc Sensor-Network Simulation Environment: H-MAS . The University of Notre Dame are simulating mobile ad-hoc sensor networks. "The purpose of H-MAS is to provide a convenient platform on which to evaluate a variety of mobile

ad-hoc sensor network configurations at the physical, medium access, network, and applications layers" [74]. This network simulator has 4 basic types of nodes: sensor nodes, processing nodes, sink nodes, and communication nodes. Sensor nodes can be further categorized based on the type of data being sensed. Signal processing, error correction, and data compression are done by the processing nodes. Sinks store the data locally until a large off network data collector is available.

H-MAS is based on the Swarm toolkit 2.1.1 [3]. The toolkit is a software package for multi-agent simulation of complex systems. The basic architecture of Swarm is the simulation of collections of concurrently interacting agents. The toolkit provides data probing, statistical analysis, an object oriented architecture, and GUI support. The source code and development environment for Swarm is available. It requires a Linux-compatible platform.

The H-MAS program is still in the "proof of concept" stage and as such is not available; however, the class structure and hierarchy are discussed in [74].

Network Simulator: ns2. The network simulator from the University of California Berkeley [65] is well known in the research community. This single-processor program is designed for use under a Linux-compatible environment. It is an event-driven network simulation program for a variety of IP networks. Because this program is open-source many users have contributed new network protocols and validated existing ones. Both wired and wireless protocols have already been defined in *ns2* libraries and can be used during simulation. A built-in visualization tool called the Network Animator provides post-processing data probing and automated analysis.

Designing a network simulation in *ns2* requires the user to know how to use the tool command language (TCL) as well as any protocol-specific parameters. The TCL is really just an easy user interface to the C++ system so that multiple (difficult) changes are not required to run a simulation. Extending *ns2* protocol details is be done through a second language (C++) because the simulator has separated the user configuration language (TCL) from the low-level simulator details. To build a simulation model, the nodes must be defined and arranged within a network topology,

any sources, sinks, and applications are then attached to the nodes, and then the simulation begins. Documentation–including several examples and tutorials–are readily available.

Objective Modular Network Testbed in C++: OMNeT++. OMNet++ authored by Andras Varga [107] is a discrete event simulation tool targeted for communication networks. The latest open source release is dated June 2003. This program is the culmination of Varga's PhD research. Similar to the dual-language sim-package *ns2*, this program uses C++ for component level definitions and NED (NEtwork Description) for assembly of components into a hierarchical network model.

This package has a corresponding GUI for the main user functions: the network topology editor, simulation execution, and graphic output vector plotting tool. There is also a command line tool for simulation execution as well as an integrated debugger. Source code for custom development is available for both Windows and Linux platforms.

There are eight advertised models that range from wired to wireless protocols including IPv6Suite. Several users have made their own contributions to the project including one called remote OMNet which uses Java RMI to manage remote simulation runs on a cluster of workstations. The website provides a comparison of OMNeT++ to other popular simulation tools such as *ns2* and GloMoSIM (`http://www.omnetpp.org/external/doc/html/usman.php#sec102`.)

*Parallelized Network Simulator (*ns2*): PDNS.* This parallelized version of the popular *ns2* came out the Georgia Institute of Technology [86]. They created PDNS to exploit scalability, parallel and distributed simulation techniques, and enable widespread use of network simulation models. This program is installed as an update to the *ns2* package described earlier. The syntax for the network topology only differs when describing packet destinations and logical connections. The implementation separates the physical connection from the logical connectivity of the network model that is being simulated through the use of IP-address abstractions.

Like *ns2* the source code is TCL for the configuration and network description and C++ for the low level protocol definitions. This also is open-source software and can be downloaded

from `http://www.cc.gatech.edu/computing/compass/pdns/`. Because of the distributed simulation environment it is necessary to implement a time manager. Georgia Tech's *RTIKIT* provides the necessary services allowing the distributed simulations to be synchronized. The latest version of this software was released June 23, 2003.

Simulator for Wireless Ad-hoc Networks: SWAN. Liu from Dartmouth College developed SWAN as part of his dissertation research [59]. The application of the simulator is detecting the status chemical, radioactive, or other catastrophic agents within a geographical region. SWAN is made up of four types of sub-models: a Terrain Model, a Plume Dispersion Model, an RF Channel Model and a Node Model.

SWAN is based on Dartmouth's Scalable Simulation Framework (DaSSF), which is "a process-oriented, conservatively synchronized parallel simulator, which is designed for simulating very large scale multi-protocol communication networks. DaSSF is a C++ implementation of Scalable Simulation Framework (SSF), with the goal of achieving scalability, manage-ability, and portability for complex simulation models" [60]. The SSF application programming interface is object oriented and defines five base classes: entity, process, outChannel, inChannel, and event. Currently the SWAN/DaSSF effort has moved to the University of Illinois in Champagne. As the development process continues a next generation DaSSF is being released under the alias iSSF, which boasts to be ultra fast and will soon be HLA compatible [61].

B.4.1.3 Simulation Frameworks. The first two categories are self-evident, however the third requires an introduction. Simulation frameworks are simulation platforms usually with a corresponding programming language built on top of a lower level language. They are designed with standards incorporated and usually well-thought out. These simulation frameworks consist of the simulation engine, associated languages, libraries, and tools to create a custom simulator. Thus beginning with a simulation framework is like starting from scratch–but there are some advantages–as noted below.

Global Mobile Information System Simulator: GloMoSIM. The University of California (Los Angeles) designed a scalable network simulation environment that they have labeled GloMoSIM [11]. It utilizes parallel execution to reduce the simulation time of detailed high-fidelity models of large communication networks at multiple layers in the protocol stack. In order to use the parallel resources, a parallel simulation language was developed as a means for implementing a GloMoSIM experiment, PARallel Simulation Environment for Complex systems (PARSEC.) The C-based PARSEC language can execute on both Windows and UNIX variants. PARSEC is designed to cleanly separate the description of the simulation model from the underlying simulation protocol used to execute it. Thus, with few modifications PARSEC programs may be executed using sequential or parallel protocols. The language is not restricted to an optimistic parallel protocol but can also run conservative protocols depending on the application.

GloMoSIM is a scalable simulation library for wireless network systems built using the PARSEC simulation environment. The authors state that GloMoSIM was designed for very large scale network simulations of sizes in the millions of nodes without significantly increasing execution times. GloMoSIM uses node aggregation in which several simulated nodes of the system are described with a single simulation entity. This is necessary for scalability. This library is available for download from `http://pcl.cs.ucla.edu/projects/domains/glomosim.html`.

High Level Architecture: HLA. According to [1], "The High Level Architecture (HLA) is a general purpose architecture for simulation reuse and interoperability. The HLA was developed under the leadership of the Defense Modeling and Simulation Office (DMSO) to support reuse and interoperability across the large numbers of different types of simulations developed and maintained by the DoD. The HLA was approved as an open standard through the Institute of Electrical and Electronic Engineers (IEEE) - IEEE Standard 1516 - in September 2000." A simulation framework is not always software as shown for HLA, but rather a specification, similar to that of CORBA. Due to the widespread mandate that all DoD simulators be compatible with HLA (in 2001 all non-HLA-compliant DoD simulators were retired) this architecture is necessar-

ily ubiquitous. That means that several large-scale simulators that were not able to communicate with each other can now communicate via the HLA defined interfaces.

HLA is an architecture for distributed interactive simulation and thus requires a runtime infrastructure (RTI.) Initially, the RTI for HLA efforts was designed by the DoD and made publicly available. Recently, however, the DoD announced the commercialization of RTI but still distributes. HLA has three defining elements: the rules, object model template, and the interface specification [100]. The rules define how the simulations cooperate. The object model template defines an object view on the simulations. The interface is the application specification interface that all simulations must comply with; and, all communication between simulations is only allowed via this interface.

A thorough review of HLA is presented by Steffan Straßburger in his dissertation [99].

Synchronous Parallel Environment for Emulation and Discrete-Event Simulation: SPEEDES. SPEEDES is a general purpose parallel discrete event simulation framework created by Metron Incorporated [4, 69]. This simulation package is available to registered non-commercial users limited to use in the United States of America. SPEEDES provides several methods of running that include conservative schemes (guarantee no events will be erroneously processed) and optimistic schemes (may need to undo or roll back events). These strategies can be used to aid the modeler in taking the greatest advantage of the hardware that is available. It is also designed to implement HLA federations of simulations.

A SPEEDES simulation is made up of C++ objects and events that act on those objects[1]. Events can act on exactly 1 object (i.e. change object state) at a time, and/or schedule event(s) for the same or different objects; an event can be coded up as a simulation object method (among other ways). SPEEDES begins by distributing the objects over N "nodes" and then calls a virtual Init() method on each simulation object (a SPEEDES node is a Unix process, so there can be 1 or more nodes per CPU). The Init() calls generate events which generate more events, etc. This process continues until the simulation end time is reached (specified in an input file). SPEEDES

[1]Gary Blank from Metron Inc. provided this paragraph description of SPEEDES.

coordinates things across the N nodes, but if you use the optimistic time management algorithm (the default), the state of your simulation objects must be "rollbackable". This means that the object can revert to an earlier state if a "straggler" event arrives. For example, if an event changes the state of objX at t=100.0, and then a "straggler" event arrives for objX with timestamp t=80.0, objX must revert (or "roll back") its state to what it was at t=80.0 in order for the straggler event to have the correct state for time t=80.0. SPEEDES provides special rollbackable state classes that automate this process.

One of the more remarkable features of this package is the documentation. A published user's guide and API manual is available for download–which includes hands-on working examples [4]. Also, after registering on the web, a personal email was sent asking if there were any ways that the developers could help. Support is excellent.

B.4.2 Analysis. In this section the three simulator groups are weighed according to the desired characteristics. This is done through noticing advantages and disadvantages within each category. Finally, a more thorough analysis is given to the outstanding simulators in each category.

B.4.2.1 Swarm Simulators. Each swarm simulator generates data representing moving objects in some coherent fashion. All contain some form of visualization to perceive the emergent behavior. Open source code is the first filter applied to the simulations. Providing the source code is the only way in which reuse can be done efficiently. Thus Lua's simulator is out. Effort is still underway to obtain the source code for the Icosystems Simulation. The next big filter is the documentation that accompanies the program source code. SWEEP is severely lacking thorough documentation and thus is on the back burner unless something shows up. On the other end are simulators with excellent documentation which are Kadrovach's swarm, Icosystem's Simulation, and AFRL's MultiUAV. The user interface–both command line and graphical–are common across remaining candidate simulators. One major drawback with MultiUAV package is the source code requirement for a licensed copy of the Matlab and Simulink tools. In addition, since the author has little experience with either of those design tools and also because of the scaling problem with MultiUAV, it is out. Icosystem's and Kadrovach's simulators are the best when consider-

Table 20 Side by side Comparison

Executable	operating system	open source	documented	source code	software design	graphical interface	command line mode	built-in statistical analysis	data monitoring/custom analysis	built-in visualization tools	environment	target	vehicle	communications	sensor	data fusion	behavior	allocate tasks	supports message passing	parallel platform	scalable	state-saving support
swarm	Win	y	y	C++	OO	y	y	-	y	y	L	-	M	H	L	-	H	-	-	-	-	-
swarm(lua)	Java	-	y	Java	OO	y	u	u	u	y	L	M	M	M	L	-	H	-	-	-	-	-
Simulation	Java	u	y	Java	OO	y	y	u	l	y	M	L	M	L	H	-	H	-	-	-	-	-
TextSwarm	Java	y	l	Java	OO	-	y	l	-	y	M	L	-	-	M	-	H	-	-	-	l	-
MultiUAV	both	y	l	m1	FD	y	y	-	-	y	L	M	H	L	M	M	M	-	y	l	l	u
SWEEP	Java	l	-	Java	OO	u	y	-	y	y	L	-	L	-	M	L	M	-	-	-	-	-
ANSim	Java	l	y	Java	OO	y	y	y	y	y	L	-	-	M	-	-	L	-	-	-	-	-
Cougar	Lin	-	l	m1	FD	y	-	-	y	y	M	M	u	L	H	L	u	-	-	-	-	-
H-MAS	both	-	l	Java	OO	y	y	y	y	y	L	-	-	M	L	L	H	-	-	-	-	y
ns2	both	y	y	m2	OO	l	y	y	y	y	-	-	-	H	-	-	-	-	-	-	-	-
OMNeT++	both	y	y	m3	OO	y	y	l	-	y	-	-	-	H	-	-	-	-	l	l	-	-
PDNS	both	y	y	m2	OO	l	y	y	y	y	-	-	-	H	-	-	-	-	y	y	y	-
SWAN	Lin	y	-	C++	OO	-	y	-	-	-	L	L	-	M	M	-	M	y	y	y	y	u
GloMoSIM	both	y	y	C	FD	-	-	-	-	-	-	-	-	H	-	-	-	y	y	y	y	u
HLA	-	-	y	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	y	y	-	-
SPEEDES	both	l	y	C++	OO	-	-	-	-	-	-	-	-	-	-	-	-	y	y	y	-	y

KEY:
(Win)- Windows Platforms; (Java)- Java Virtual Machine Plaftorms; (Lin): Linux Platforms; (both): Windows and Linux
(OO) Object Oriented; (FD) Functional Decomposition
(H) high fidelity model; (M) medium fidelity; (L) low fidelity
(l) supported but limited
(y) supported in full
(u) unknown
(-) not supported
(m1) miscellaneous: Matlab, Simulink, C++
(m2) miscellaneous: TCL, C++
(m3) miscellaneous: NED, C++

ing the completeness and fidelity of the seven desired models, thus places Lotspeich's TextSwarm behind. Noteworthy, however, is Lotspeich's path-planning model. Since neither of the top two simulators (swarm, Simulation) include high performance computing constructs then they are tied for the best swarm simulator. One distinguishing attribute is that Kadrovach's communications model is not original, rather it plugs in data to an existing network simuator so that there is extra overhead during the translation process. Nonetheless, the best swarm simulator based on the stated criteria is a tie between Icosystem's and Kadrovach's simulators.

B.4.2.2 Network Simulators. While the visual display is a critical aspect when considering swarm simulators it is not when analyzing network simulators. Often network simulations produced metrics that are easily visualized with a graphing program; however all but one of the network simulators include some form of visualization. Cougar and H-MAS break the open source requirement. While SWAN has many exciting features to include parallel development framework the documentation and program maturity is lacking. The critical filter for network simulators after documentation and open source characteristics is the data analysis capacity. OMNeT++ has the advantage when it comes to ease of use for its many GUI interfaces and development environments. It also includes an excellent integrated debugging tool unlike all other network simulators considered for this review. The one major drawback is its limited built-in analysis capability. When coupled with the lack of parallel support, OMNeT++ is out. ANSim when set beside *ns2* loses on all fronts except ad-hoc wireless networks for connecting two randomly connected stations. ANSim is developed toward a specific objective–as expected from a research student. *ns2* on the other hand has not only ad-hoc wireless network models but also wired protocols, transmitter ranging resolutions, packet level analysis, validation tests, and many more model fidelity options due to its widespread acceptance in the networks research community. PDNS is simply *ns2* forced into the parallel processing world. Unfortunately, PDNS requires that the scaled network be designed with border-routing protocols which does not work well with swarming applications. Although *ns2* lacks a parallel interface it is the best network simulation based on the stated criteria.

B.4.2.3 Simulation Frameworks. Common to all simulation frameworks is the main disadvantage that all models must be coded from the ground up–reuse is limited to others who have used the same simulation framework for a similar design–which is rare. Advantages of GloMoSIM–based on the PARSEC language– include excellent documentation and a high level of maturity (relatively large user base over several years.) The disadvantages with this framework are its *new* language–requires learning a variant of C–and lack of object-oriented capability, thus GloMoSIM is out. HLA is a candidate purely for its widespread interoperability mandate for its use in the DoD's modeling and simulation community. It is a specification like CORBA and therefore requires a third party to derive an implementation just for each simulation interface–in addition to the model. Some implementors have taken thousands of lines of code for HLA compliance and use an entire suite of development tools [2]. On the positive side, HLA allows multiple heterogeneous simulations to interact using a runtime infrastructure for synchronization. The advantages of the SPEEDES framework are its excellent documentation, state-saving features, highly configurable synchronization schemes, and object oriented base. The built-in parallel constructs allow one to quickly parallelize programs. Thus SPEEDES is the simulation framework of choice.

B.4.2.4 In-depth Review. The best candidates from the above categories are now given a further look. Because of the diversity of categories only some of the desired characteristics apply to each candidate. As such the in-depth review expands on those relevant elements of the simulator. The top four programs are swarm, Simulation, *ns2*, and SPEEDES. Table 21 presents a lower level of detail comparison based on the user's experience and program documentation. The supported features are now rated on a scale from 1 to 5, with 1 representing the author's least value and 5 representing the author's best value selection.

swarm (Kadrovach). The details in this section are from Chapter 4 of [53]. While swarm is based on Microsoft's platform development kit for the GUI version, it is conceivable that the command line mode of the program can easily be ported to a Linux version and then enhanced for parallelization. The C++ code base has a large performance advantage over the same program written in Java. After several experiences with the GUI version of swarm it

Table 21 Final Candidates Side by Side

Executable	operating system	open source	documented	source code	software design	graphical interface	command line mode	built-in statistical analysis	data monitoring/custom analysis	built-in visualization tools	environment	target	vehicle	communications	sensor	data fusion	behavior	allocate tasks	supports message passing	parallel platform	scalable	state-saving support
swarm	Win	5	4	C++	OO	3	5	-	2	2	L	-	M	H	L	-	H	-	-	-	-	-
Simulation	Java	u	3	Java	OO	4	5	-	3	5	M	L	M	L	H	-	H	-	-	-	-	-
ns2	both	5	2	m2	OO	-	5	3	3	3	-	-	-	H	-	-	-	-	-	-	-	-
SPEEDES	both	5	4	C++	OO	-	-	-	-	-	-	-	-	-	-	-	-	3	4	5	5	3

KEY:
(Win)- Windows Platforms; (Java)- Java Virtual Machine Plaftorms; (Lin): Linux Platforms; (both): Windows and Linux
(OO) Object Oriented; (FD) Functional Decomposition
(H) high fidelity model; (M) medium fidelity; (L) low fidelity
(l) supported but limited
(1 -5) scale of value; 1- least, 5- best
(u) unknown
(-) not supported
(m2) miscellaneous: TCL, C++

became evident that only a correct operation sequence can make the simulator work–the GUI is unstable. This shortcoming pertains solely to the visualization portion of the program and thus can be overcome by using Matlab as the visualization platform as Kadrovach has supplied several visualization examples in Matlab format.

To understand the quantitative values of the fidelity assignment in the side by side comparison this paragraph will explain it. The low fidelity environment model includes swarm orientation information in a 2D coordinate format along with a direction angle. No other environmental affects are incorporated. The high fidelity behavior model includes influences such as neighbor position and velocity, boundaries, and way points. It is a decentralized algorithm that calculates the weighted influence of attraction, alignment, and update vectors. Other influences are detailed to mimic real life aspects swarms such as birds. In particular a visibility characteristic is built into the vehicle model. This visibility is also taken into account in the behavior model. In addition the vehicle model includes aspects such as maximum speed, turning radius, and separation distance (or vehicle dimensions.) The neighbors of a particular particle are also weighted according to the region in which they reside, for example the closer the neighbor is the larger the influence on the swarm behavior. Finally, the sensor and communications models listed with the swarm program are really indirect references to the high-fidelity network simulator, *ns2*. What Kadrovach provides is an interface that translates the position information into a network topology for a wireless ad hoc network. He primarily uses the Directed Diffusion protocol and point sources with limited transmission ranges to model the swarm communications network.

Simulation (Icosystems). The Simulation swarm program included an initial and extended version as part of the Phase I contract. The details in this section are from [29, 41]. The selection of Java as the programming language is a good choice for portability but poor when considering efficiency. Nonetheless, the object-oriented structure is easily ported to the C++ language. The data monitoring capability that is part of this package includes metric outputs such as the percentage of territory covered, number of targets killed, and UAV specific coverage. The

built-in visualization tools are excellent ways of perceiving the progress of the swarm mission both in 2D and 3D.

The initial development focused on a model to study the behavior of a UAV swarm carrying out an area search mission using a percentage of coverage metric for evaluation. The extension allows UAVs to track and attack targets. The terrain model is defined as a rectangular region which is further subdivided into a grid used primarily to determine the coverage and track pheromone signals. The vehicle model can fly at variable speeds and altitudes with independent pitch and yaw control. The control dynamics are simplified by specification of a maximum turn rate and the ability to change the amount of thrust. Several sensors are modeled on each UAV: forward cone-shaped, circular, GPS, pheromone, and boundary detector. The forward-looking ground sensor has adjustable radius and angular aperture for detecting terrain and targets. It is stochastically modeled based on the distance, elevation, and amount of time spent flying over a given terrain cell. The circular sensor detects the presence of other UAVs within an adjustable radius. The GPS determines position. The pheromone sensor allows each UAV to detect within a small rectangular region centered on itself how much each terrain cell has been covered. The pheromone dissipates over time. The boundary sensor indicates the end of the terrain and is used to contain UAVs in the target region. Targets can be either static or dynamic and are randomly distributed. A target has two states: alive or dead. The most favorable UAV control strategy in this simulation is a combination of using pheromones, repulsion, and jitter strategies. Curiously lacking detail in the reports is a communications model–however it may be understood that the pheromones are the indirect communications model being used.

In addition to the swarm simulation a genetic algorithm was developed to tune the parameters for best performance based on the mission. Details of this algorithm can be found in [29].

ns2 *(University of California Berkeley).* The detailed information presented on the *ns2* simulation program came from [27]. References to NS is synonymous to *ns2*.

The *ns2* single-processor program was written for the Linux environment with the intent of using add-on packages and custom builds to provide the desired network simulation. This code is

a derivative of the REAL network simulator and has evolved ever since 1989 [65]. *ns2* depends on several externally available software components including: Tcl/Tk, otcl, TclCL, and many more. The following are optional but very useful: nam-1, xgraph, perl, tcl-debug, dmalloc, sgb2ns, tiers2ns, and Cweb. It is obvious that this open source project exceeds several thousand lines of code and hence is in a class of software programs at the enterprise level.

The interface to the network simulator consists majorly of three types of output. Two of the outputs are textual representations of what is occurring during the simulation so these forms require further analysis by the user. One of these outputs are to the standard output device on the platform the user is running. This can be either the monitor or some other output device such as a storage device or a port device. This output contains certain items of interest such as setup details, and any fail terms. The user sets up this output in the TCL file via output statements. The second output is to a trace file created by the NS simulator. The implementation of the trace file is not standard though, and must be implemented in the TCL script file the user is running in order to receive this output. This output consists of data on the transmission lines between the different communication devices such as bandwidth usage. This data can be represented in a graphing program incorporated with the NS package. The last output device is an animation file which must also be declared in the TCL file in order to receive this output. This shows the actual transmission of data in a graphical form in order to give the user a better idea of what is going on. The latter two outputs can be represented in a GUI with a few modification devices in these menus.

Modification on the NS package is easy, once understanding of the TCL and C++ files is complete. Since the source files are included with the package, modification is as easy as modifying one of the source files and recompiling everything. However, one must first understand the nature of the interaction between the C++ files and the TCL files. There is an interface, which this paper will not go into, which exists that requires modification in both the C++ and the TCL environment in order to get NS to run properly. The NS package does include decent documentation along with a user's manual, although the manual is three hundred eighty pages of difficult reading material.

Everything in the NS world is represented as an object, with data members for that object and functions which act on the object. It includes inheritance for multiple classes of objects organized

into a hierarchical structure. The TCL models this structure in its own environment and C++ has objects which create the interface between these two environments.

The high-fidelity model available from *ns2* is a network model. It has the following protocol levels:

- wired, wireless, satellite
- TCP, UDP, multicast, unicast
- web, telnet, ftp
- ad hoc routing, sensor networks

The infrastructure includes stats, tracing, error models, simple trace analysis, often in Awk, Perl, or Tcl. The available routing and queuing includes wired routing, ad hoc routing and directed diffusion. The available queuing protocols include: drop-tail, RED, fair queuing, CBQ, FQ, SFQ, DRR. There are two types of nodes in NS as are described below.

Unicast node has an address classifier that does unicast routing and a port classifier

Multicast node has a classifier that classify multicast packets from unicast packets and a multicast classifier that performs multicast routing

Another fundamental object in *ns2* is a link. It is a major compound object in NS. When user creates a link using duplex-link member function, two simplex links in both directions are created. It is used to connect nodes and complete the topology in *ns2*. There are also Pareto On/Off Traffic Generator in which packets of information are sent at a fixed rate during on periods and no packets are sent during off periods.

A key feature of the wireless models is the MobileNode object which consists of the MobileNode at the core with additional supporting features that allows simulations of multi-hop ad-hoc networks, wireless LANs etc. MobileNode object is a split object. The C++ class MobileNode is derived from parent class Node. The MobileNode thus is the basic Node object with added functionalities of a wireless and mobile node like ability to move within a given topology, ability

to receive and transmit signals to and from a wireless channel. However, the difference between Node and MobileNode is that a MobileNode is not connected by means of Links to other nodes. The mobility features including node movement, periodic position updates, maintaining topology boundary are implemented in C++ while plumbing of network components within MobileNode itself (like classifiers, dmux, LL, Mac, Channel etc) have been implemented in Otcl.

SPEEDES (Metron Inc.) This simulation framework is thoroughly documented in [69]. Because this is a development framework for parallel discrete event simulation it has no GUI but rather libraries that are used in the development of a parallel simulation. This framework is implemented in C++ and thus is both efficient and object oriented. The framework started development in the early 90's and has since been continuously monitored and improved upon. While this framework has been around for over a decade the main user base is government-sponsored, for example the following are users: Joint Modelling and Simulation System, Joint SImulation System, the Extended Air Defense Test Bed, the Defense Modeling and Simulation Office Knowledge Framework Test Bed, and Joint National Test Facility's Wargame 2000 [31].

The provided functionality solves a common problem among all parallel discrete event simulators: synchronization. The fundamental challenge is to efficiently process events concurrently on multiple processors while preserving causality of the system as it advances in simulated time. SPEEDES allows the user to specify at runtime whether to use a conservative, optimistic, or configuration somewhere on the continuum in the middle via the run-time parameters. Using a conservative approach places several limitations on how simulated objects interact. Using an optimistic routine however, allows for more liberty in the design. SPEEDES uses a patented "Breathing Time Warp" synchronization algorithm which prevents instability from the risk of cascading rollbacks (optimistic) and avoids too many synchronizations (conservative.) See Appendix A in [69] for more detail.

Several run-time parameters specify the configuration that SPEEDES executes. While this parameter file can be used to customize the synchronization of the simulation, it is optional, at which point default values are used. Some samples of the types of user-configurable data in this

parameter file include the synchronization mode, number of nodes, simulation end time, lookahead, and global virtual time parameters to name a few. In addition to the built-in HLA support there is also a built-in external module class. This class, SpStateMgr, receives "committed" or "released" events from the simulation for use.

B.4.3 Selection . From the detailed analysis of the top four candidates it is apparent not one simulator has the focused traits of interest across all desired characteristics and thus selection of a single simulator platform requires relaxing the requirements in one or more categories. Another approach to this selection process is to select a framework from which to build an aggregate solution that fulfills all desired characteristics listed in Table 17. This approach allows for flexibility in choosing not just one simulator because of a high-fidelity model, but also provides for reuse across swarm simulations that were originally eliminated, assuming the source code is available and translatable.

The selection is now a layered approach in which a base system serves as the foundation upon which all other simulation components are integrated. Since the lower the level the higher the impact of parallel computing performance, the bottom layer is a good place to use the parallel infrastructure, so that SPEEDES is the ideal solution. Because SPEEDES already has the necessary constructs and run-time infrastructure to execute a PDES it is an excellent bottom layer. The next layer are the component applications that allow for the top layer Swarm Reconnaissance simulation. It must include all the models mentioned in this paper. Since the top performers in the swarm simulators include Kadrovach and Icosystems simulators they are the first simulators to be added in the component applications layers. Choosing which model from which simulator is now an easier decision since SPEEDES is implemented in C++; other simulations which also have the same language are first on the priority list. Kadrovach's swarm simulator and the communications simulator *ns2* are both native C++ applications and therefore are logical starting points for inclusion into the component application layer. While the target, sensor, and environment models have yet to be chosen several candidates remain. One subtlety not yet mentioned is that the fusion model

unfortunately has only one candidate to choose from. Finding an appropriate fusion fidelity model for the Swarm Reconnaissance application is the subject of future research.

B.5 Conclusion

This paper reviewed the problem of simulating a Swarm Reconnaissance mission including such aspects of the communications, vehicle movements, sensor characteristics and fusion, swarming behavior, target descriptions, and environment models. Three categories of simulators were evaluated against the desired set of characteristics: swarm simulators, network simulators, and simulation frameworks. It was found that none of the simulators encapsulated all of the desired simulation traits. This led to the decision to use a layered selection approach where the SPEEDES parallel environment was chosen as the lowest layer and the initial two component models to be selected are Kadrovach's swarm behavior model and the network simulator, *ns2*. As the implementation progresses future work includes selecting other models for integration into the SPEEDES framework. While finding an all-in-one swarm simulator for the characteristics described in this paper did not happen, this effort has served as a launching point toward arriving at a comprehensive Reconnaissance swarm parallel discrete event simulator.

Appendix C. Additional Data

The data contained herein is additional support for the testing and analysis conducted in this research effort.

C.1 Experiment P1: Parameter Input Files

C.1.1 Params.txt.

```
Region 3000 2000
Popsize 15
Scale 50.0
CZone 10.0
Dir 0.0
Seed 5216
Type 1
Velocity 1.0
Turn 5.0
```

C.1.2 Swarm.dyn.

```
; Lines that begin with a semi-colon are ignored
;
; Parameters:                          (typical values)
;   A     - Boundary repulse weight        30.0
;   B     - Periph weight                  10.0
;   C     - Waypoint weight                20.0
;   D     - Repulse weight                 30.0
;   E     - Alignment weight                0.5
;   F     - Linear bias (slope)             0.5
;   G     - Attraction weight               1.0
;   N_h   - Neighborhood size (int)         7
;   v_vac - velocity factor                 0.2
```

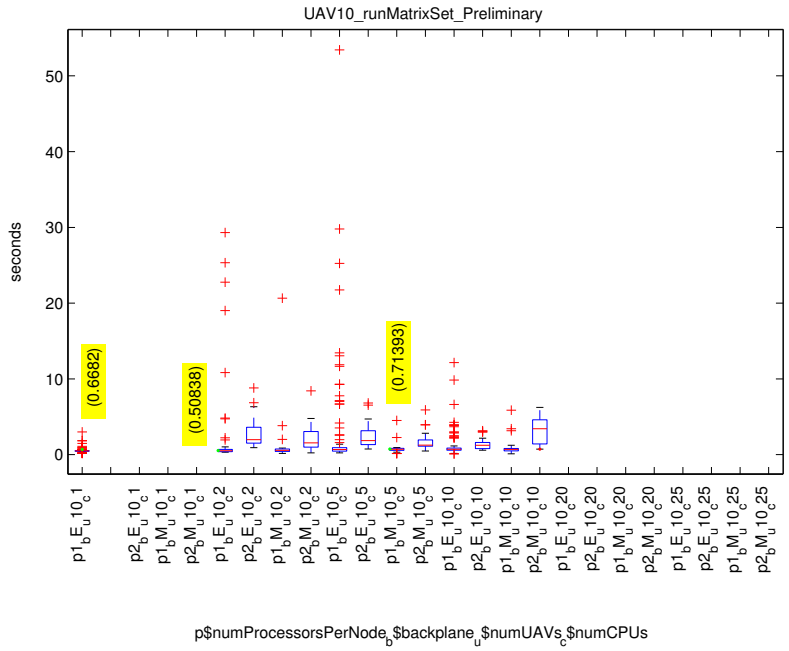


Figure 33 Box plot of Elapsed Time Data for Experiment 1 for 10 UAVs

```
;    t_max - max turn amount (degrees)        5.0
;    czone - comfort zone [0, 1]              0.1
;    p_max - max turn perturb                 2.0
;
;     t      A     B     C     D    E    F     G    v_fac t_max czone p_max
;------- ---- ---- ---- ---- --- --- ----  ----- ----- ----- -----
;
    0.000 10.0  1.0 20.0  8.0 8.0 0.0  6.0  0.008   4.0   0.1   4.0
  500.000 10.0  1.0 20.0 24.0 0.0 0.0 12.0  0.012   4.0   0.9   4.0
; End of file, file must end with at least one comment line
```

C.2 Experiment S1: Box plots of Preliminary Data

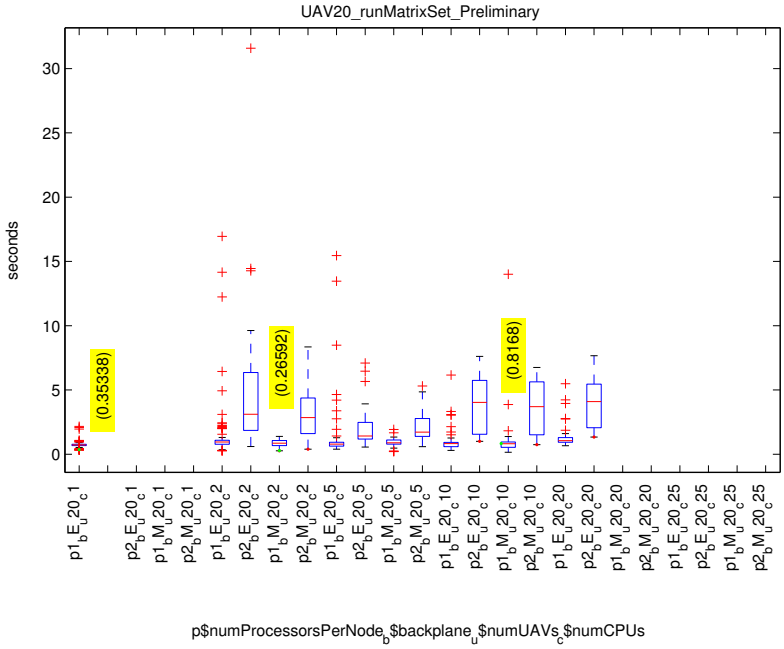


Figure 34 Box plot of Elapsed Time Data for Experiment 1 for 20 UAVs

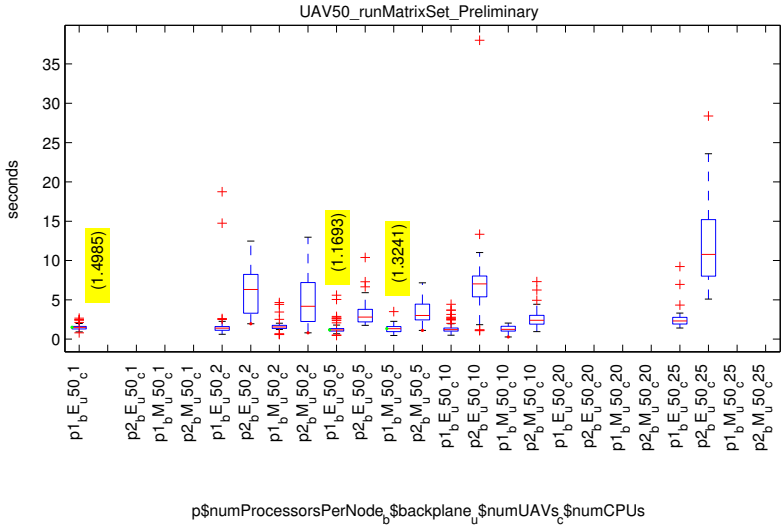


Figure 35 Box plot of Elapsed Time Data for Experiment 1 for 50 UAVs

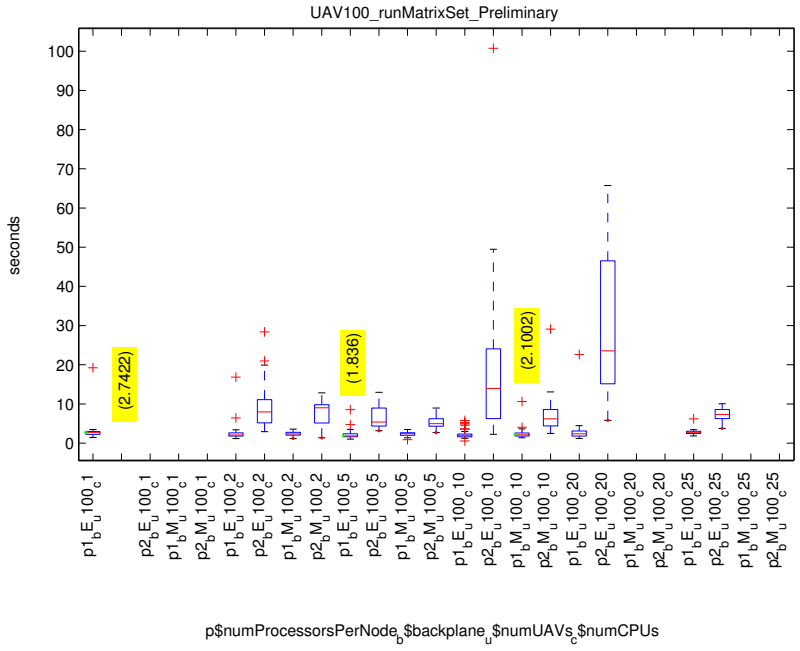


Figure 36 Box plot of Elapsed Time Data for Experiment 1 for 100 UAVs

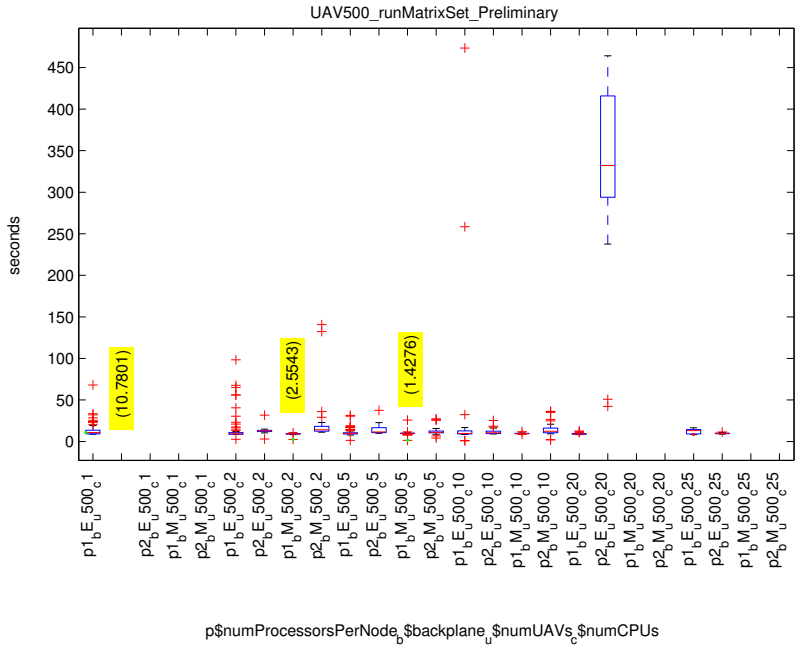


Figure 37 Box plot of Elapsed Time Data for Experiment 1 for 500 UAVs

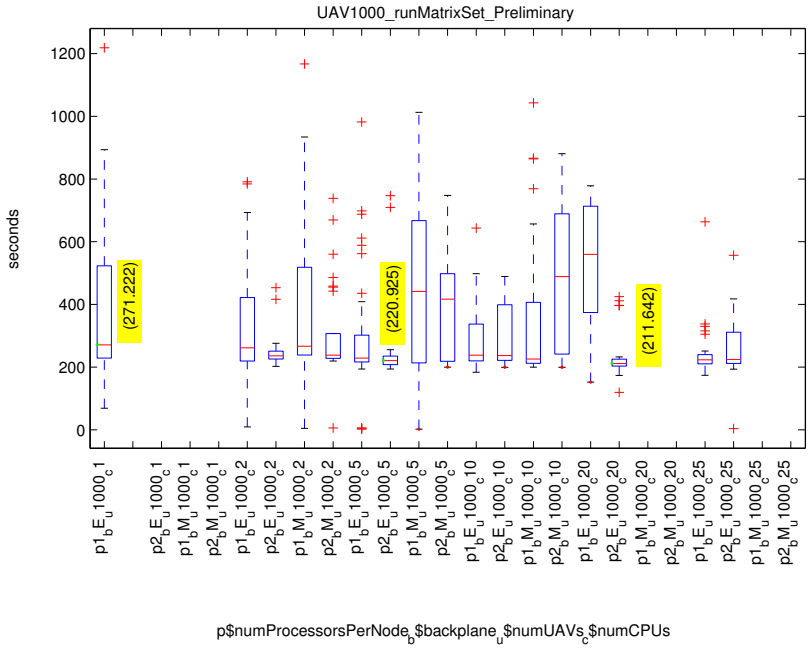


Figure 38 Box plot of Elapsed Time Data for Experiment 1 for 1000 UAVs

Bibliography

1. "Defense Modeling and Simulation Office." https://www.dmso.mil/public/transition/hla/.

2. "LabWorks, The Tools of Choice for Supporting the HLA Lifecycle." www.aegistg.com/labcut/Labworkscut.html.

3. "Swarm Development Group." www.swarm.org.

4. "Synchronous Parallel Environment for Emulation and Discrete-Event Simulation." http://www.speedes.com/index.html.

5. "Doctrine for Reconnaissance, Surveillance, and Target Acquisition Support for Joint Operations." Joint Pub 3-55, April 1993. Joint Publication of the United States Army, Navy, Air Force, and Marines.

6. "Cavalry Troop." Headquarters, Department of the Army, October 1995. FM 17-97.

7. "Air Reconnaissance," July 2003. MCWP 3-26; United States Marine Corps; Department of the Navy.

8. AFRL/VACA, "MultiUAV Simulation User Documentation Version 1.1," October 2001.

9. Aspen Systems, Inc., "Aspen Systems - High Performance Technical Computing Specialists." http://www.aspsys.com/.

10. Bagrodia, Rajive. "Perils and Pitfalls of Parallel Discrete-Event Simulation." *Winter Simulation Conference*. 136–143. 1996.

11. Bajaj, Lokesh, et al. *GloMoSim: A Scalable Network Simulation Environment*. Technical Report 990027, 1999. citeseer.nj.nec.com/225197.html.

12. Banks, Jerry, et al. *Discrete-Event System Simulation*, chapter 1,3, 3, 10, 12, 64–65. Prentice Hall, 2001.

13. Bone, Elizabeth and Christopher Bolkcom, "Unmanned Aerial Vehicles: Background and Issues for Congress." Report for Congress, April 2003. p.9 (five major).

14. Brown, David A., “Medusa’s Mirror: Stepping Forward to Look Back "Future UAV Design Implications From the 21st Century Battlefield",” 1998. School of Advanced Military Studies; United States Army Command and General Staff College.

15. Brown, Robert G., et al., “Beowulf mailing list FAQ, version 2,” 1999. Available from: `http://www.canonical.org/kragen/beowulf-faq.txt`.

16. Bruegge, Bernd and Allen A. Dutoit. *Object-Oriented software engineering; conquering complex and changing systems*. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

17. Burden, Richard L. and J. Douglas Faires. *Numerical Analysis Seventh Edition*. Brooks/Cole, 2001.

18. Bush, George W., “President Speaks On War Effort To Citadel Cadets,” December 2001. Remarks by the President; `http://www.whitehouse.gov/news/releases/2001/12/20011211-6.html`.

19. Buyya, Rajkumar. *High Performance Cluster Computing*, *1*. Prentice Hall, 1999.

20. Carmichael, Bruce W., et al., “Strikestar 2025,” August 1996. Air War College Publication; Air Command and Staff College, Maxwell AFB, AL.

21. Cassinis, R., et al. “Strategies for Navigation of Robot Swarms to be used in Landmines Detection.” *Proceeding of the Third European Workshop on Advanced Mobile Robots*. 1999.

22. Clough, Bruce. “UAV Swarming? So What Are Those Swarms, What Are The Implications, And How Do We Handle Them?.” *Proceedings of 3rd Annual Conference on Future Unmanned Vehicles*. April 2003. Air Force Research Laboratory, Control Automation.

23. Company, Javvin, “Fast Ethernet: 100Mbps Ethernet (IEEE 802.3u) Overview,” 2003. `http://www.javvin.com/protocolFastE.html`.

24. Cooper, Mendel, “Advanced Bash-Scripting Guide, an in-depth exploration of the art of shell scripting,” June 2003. Version 1.8.4; `http://www.faqs.org/docs/abs/HTML/index.html`.

25. Corner, Joshua, “Beowulf Configuration for Large-Scale Generic Application,” 2003.

26. Corner, Joshua, “Master’s Thesis Proposal,” September 2003.

27. Corner, Joshua, et al., "Case Study: Parallel NS Simulator," May 2003. CSCE 656 Final Project Report.

28. Corner, Joshua J., "Parallelizing UAV Swarm Communication Simulation," July 2003. CSCE790 Final Project Report, AFIT.

29. Corporation, Icosystem. *Evolutionary Swarm Intelligence for the Control of Unmanned Air Vehicles*. Technical Report 0001AE, Icosystem Corporation, March 2003.

30. Davis, C. R., "Airborne Reconnaissance: The Leveraging Tool For Our Future Strategy." Executive Research Project, National Defense University, 1995. The Industrial College of the Armed Forces.

31. Driscoll, Jonathan, et al., "Demonstrating an HLA Capability for JMASS Using SPEEDES." Huntsville Simulation Conference, HSC 2001, 2001. `https://rsic.redstone.army.mil/hsc/abstracts/040.pdf`.

32. Dudek, Gregory, et al., "A Taxonomy for Multi-Agent Robotics," 1996.

33. Felderman, R. E. and L. Kleinrock. "An Upper Bound on the Improvement of Asynchronous versus Synchronous Distributed Processing.." *Proc. of the SCS Multiconf. on Distributed Simulation22*. 1990.

34. Ferscha, Alois and Satish K. Tripathi. *Parallel and Distributed Simulation of Discrete Event Systems*. Technical Report CS-TR-3336, 1994.

35. Fields, MaryAnne and Bailey T. Haug. *Developing a Chemical Reconnaissance Behavior for Unmanned Ground Vehicles Using the OneSAF Battlefield Simulation Tool*. Technical Report ARL-TR-2972, Army Research Laboratory, 2003.

36. for CSCE790 course at AFIT; 48 page document; see Section 4, Handout, "Solving Partial Differential Equations on Parallel Computers." Gary Lamont.

37. French, Matthew, "UAVs advance since Desert Storm," February 2003. `http://www.fcw.com/fcw/articles/2003/0217/web-uav-02-19-03.asp`.

38. Fujimoto, R. M. "Parallel Discrete Event Simulation." *Communications of the ACM 33*, 30–53, October 1990.

39. Gaffney, Timothy, "Bush budget boosts Wright-Pat work," February 2004. Dayton Daily News, Business Section.

40. Garrison, Peter, "Microspies: A web anthology-a supplement to Air & Space magazine," April/May 2000. Air & Space Magazine; http://www.airspacemag.com/asm/mag/supp/am00/uSPY.html.

41. Gaudiano, Paolo, et al. "Swarm Intelligence: a new C2 paradigm with an application to control of swarms of UAVs." *8th ICCRTS Command and Control Research and Technology Symposium*. 2003.

42. Gill, Andrew W., et al., "Using ISAAC to explore the impact of reconnaissance on mission success." Defense Science and Technlogy Organisation, 2002. http://www.mcwl.usmc.mil/divisions/albert/research/documents/mws02.pdf.

43. Grama, Ananth, et al. *Introduction to Parallel Computing*. Harlow England: Addison-Wesley, 2003.

44. Grama, Ananth, et al. *Introduction to Parallel Computing, 2nd Edition*. none, Harlow, England: Addison Wesley, 2003.

45. Grant, Rebecca. "The Bekaa Valley War," *Air Force Magazine*, 58–62 (June 2002).

46. Gregory M. Nielson, Hans hagen, Heinrich Muller. *Scientific Visualization, Overviews, Methodologies, and Techniques*. none, Los Alamitos, California: IEEE Computer Society, 1997.

47. Hellbruck, Horst, "ANSim - Ad-Hoc Network Simulation Tool." http://www.i-u.de/schools/hellbrueck/ansim/.

48. Hong, Xiaoyan, et al. "Scalable Routing Protocols for Mobile Ad Hoc Networks," *IEEE Network*, 11–21 (July 2002). University of California at Los Angeles.

49. Intanagonwiwat, Chalermek, et al., "Directed diffusion: a scalable and robust communication paradigm for sensor networks," 2000. pp. 56-67; Mobile Computing and Networking; http://citeseer.nj.nec.com/article/intanagonwiwat00directed.html.

50. Iwaarden, Ronald Van, "Overview of SPEEDES," April 2003. Metron; High Performance Computing Division.

51. Jefferson, D. and H. Sowizral. "Fast Concurrent Simulation Using the Time Warp Mechanism." *Distributed Simulation 1985*, edited by P. Reynolds. 63–69. 1985.

52. Jones, James Patton, et al. *Portable Batch System User Guide*. Veridian Systems, November 2001.

53. Kadrovach, B. Anthony, "Design and Analysis of a Communications Model for Swarm-based Sensors." dissertation–unpublished as of the date of this paper; United States Air Force Institute of Technology".

54. Kadrovach, B. Anthony and Gary B. Lamont. *A Communications Modeling System for Swarm-based Sensor Networks*. PhD dissertation, Air Force Institute of Technology, 2003.

55. Kadrovach, B. Anthony, et al., "An Ad hoc Network System for Swarm-based Sensors," 2002. Mobile Computing and Communications Review, Volume 1, Number 2.

56. Kovacina, Michael A., et al. *Multi-Agent Algorithms for chemical cloud detection and mapping using unmanned air vehicles*. Technical Report AFRL-VA-WP-TP-2002-322, Air Force Research Laboratory, Air Vehicles Directorate, September 2002. Proceedings for 2002 Conference on Intelligent Robots and Systems Presentation, Lausanne, Switzerland.

57. Kumar, V., et al. *Introduction to Parallel Computing. Design and Analysis of Algorithms.*. Benjamin/Cummings, 1994.

58. Lim, Alvin. "Support for Reliability in Self-Organizing Sensor Networks," *ISIF*, 974–975 (2002).

59. Liu, J., et al. "Simulation modeling of large-scale ad-hoc sensor networks." *Proceedings of the 2001 Simulation Interoperability Workshop. London, England.*. 2001.

60. Liu, Jason, "Dartmouth SSF (DaSSF) Home Page." `http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/`.

61. Liu, Jason, "iSSF." `http://www.crhc.uiuc.edu/~jasonliu/projects/issf/index.html`.

62. Lotspeich, James, "Distributed Control of a Swarm of Autonomous Unmanned Aerial Vehicles," 2003.

63. Lua, Chin A., et al., "Synchronized Multi-point attack by autonomous reactive vehicles with simple local communication," 2002. www.cs.ndsu.nodak.edu/~lua.

64. Mataric, Maja J. "Issues and approaches in the design of collective autonomous agents," *Robotics and Autonomous Systems*, *16*(2-4):321–331 (December 1995).

65. McCanne, S. and S. Floyd, "The LBNL network simulator." Software on-line http://www.isi.edu/nsnam/ns/".

66. McDonald, Bruce. "A Proposed Process For Defining Requireed Fidelity of Simulations." *98 Spring Simulation INteroperability Workshop*. March 1998. Paper 245.

67. McHale, John. "Unmanned Aircraft armed and dangerous," *Military and Aerospace Electronics*, *14*(11):18–25 (2003).

68. McLain, Timothy W, "Coordinated Control of Unmanned Air Vehicles," 1999. Air Vehicles Directorate, Wright-Patterson Air Force Base, Ohio.

69. Metron Inc. *SPEEDES User Guide*, 2003.

70. Michael H. Dickinson, DARPA, "Biological Insight to Flight Control," 1999. http://www.darpa.mil/dso/thrust/biosci/cbs/ucb-v_ab.html.

71. Michelson, Robert, "Entomopter Project," 2002. http://avdil.gtri.gatech.edu/RCM/RCM/Entomopter/EntomopterProject.html.

72. Milton, J. S. and Jesse C. Arnold. *Introduction to Probability and Statistics, 3rd Edition*. New York, New York: McGraw-Hill Primis Custom Publishing, 2002.

73. Misra, Jayadev. "Distributed Discrete-Event Simulation," *Computing Surveys*, *18*(1):39 – 65 (March 1986).

74. Mochocki, Bren C. and Gregory R. Madey. "H-MAS: A Heterogeneous, Mobile, Ad-hoc Sensor - Network Simulation Environment." *Seventh Annual Swarm Researchers Meeting, SwarmFest*. 2003.

75. Munson, Kenneth. *Jane's unmanned aerial vehicles and targets*. Jane's Information Group, 2000.

76. Myricom, "Myrinet Overview," 2002. Available from: http://www.myri.com/myrinet/overview/.

77. Newswire, JCN, "Epson Develops World's Smallest Flying Microrobot," 2003. www.japancorp.net/Article.asp?Art_ID=5967.

78. Niepert, Robert, "Hot Air Balloons In The Civil War," 2003. See http://www.floridareenactorsonline.com/baloons.htm.

79. Organization, International Standards, "Basic Reference Model of Open Distributed Processing, Part 1: Overview and guide to use," 1992. ISO/IEC JTC1/SC212/WG7 CD 10746-1.

80. Palmer, D. W., et al. "Impact of Behavior Influence on Decentralized Control Strategies for Swarms of Simple, Autonomous, Mobile Agents." *Biomechanics Meets Robotics Modelling and Simulation of Motion*. 54–70. 1999. citeseer.nj.nec.com/455170.html.

81. Passino, Kevin, et al. "Cooperative Control for Autonomous Air Vehicles.".

82. Pike, John and Steven Aftergood, "Federation of American Scientists: Unmanned Aerial Vehicles," March 2003. http://www.fas.org/irp/program/collect/uav.htm.

83. Qi, Hairong, et al. "Distributed sensor networks–a review of recent research," *Journal of the Franklin Institute*, (338):655–668 (2001).

84. Reynolds, Craig, "Boids(Flocks, Herds, and Schools: a Distributed Behavior Model)," 2001. http://www.red3d.com/cwr/boids/.

85. Reynolds, Craig W. "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics*, *21*(4):25–34 (1987).

86. Riley, George F., et al., "A Generic Framework for Parallelization of Network Simulations." Georgia Institute of Technology. College of Computing, Atlanta GA.

87. Rogowitz, Bernice E. and Lloyd A. Treinish, "How NOT to Lie with Visualization."

88. Roza, Manfred, et al. "A Fidelity Management Process Overlay onto the FEDEP Model." *1998 Fall Simulation Interoperability Workshop*. September 1998. Paper 083.

89. Samson, Victoria. "Q & A on the Use of Predator in Operation Enduring Freedom," *The Center for Defense Information* (February 2002). http://www.cdi.org/terrorism/predator.cfm.

90. Schricker, Bradely C., et al. “Fidelity Evaluation Framework.” *34th Annual Simulation Symposium (SS01)*. 109–116. 2001.

91. Shaffer, Clifford. *A Practical Introduction to Data Structures and Algorithm Analysis*, chapter 2, 35–37. Computer Science and Applied Mathematics, Prentice Hall, 2001.

92. Simulation, I., “Proposed IEEE Standard Draft for Information Technology- Protocols for Distributed Interactive Simulation Applications,” 1993.

93. Squatriglia, Chuck, “Soaring achievement in spying UC Berkeley team creating ’microfly’ to infiltrate enemies.” Ohio State University Transgenics in the News Publication, June 2002. `http://ipm.osu.edu/trans/062_241.htm`.

94. S.S. Iyengar, L. Prasad, Hla Min. *Advances in Distributed Sensor Integration Application and Theory*. Series on Environmental and Intelligent Manufacturing Systems, Prentice hall, 1995.

95. Staff, AUVSI, “US Military Robots Employed in Iraqi War,” May 2003. /url-http://www.auvsi.org/iraq/index.cfm.

96. Steinman, Jeff. “SPEEDES: A Unified Approach to Parallel Simulation.” *Proceedings of 6th Workshop on Parallel and Distributed Simulation*. 75–83. 1992.

97. Steinman, Jeff S. “Breathing Time Warp.” *Workshop on Parallel and Distributed Simulation*. 109–118. 1993.

98. Steinman, Jeff S. “Incremental State Saving in SPEEDES using C++.” *Proceedings of the 1993 Winter Simulation Conference*. 687–696. 1993.

99. Straßburger, Steffan. *Distributed Simulation Based on the High Level Architecture in Civilian Application Domains*. PhD dissertation, Otto-von-Guericke-Universitat Magdeburg, 1999.

100. Straßburger, Steffen and Ulrich Klein, “Introduction to the High Level Architecture for Modeling and Simulation (HLA),” 1999. HLA Tutorial.

101. Strukel, Steven E., et al. *Quantifying the Value of Reconnaissance*. Technical FY93/93-7, West Point, New York: Operations Research Center, United States Military Academy, September 1993.

102. Sulistio, Anthony, et al. "A Taxonomy of Computer-based Simulations and its Mapping to Parallel and Distributed Systems Simulation Tools." *International Journal of Software: Practice and Experience*. 2003.

103. Trahan, Michael W., et al., editors. *Swarms of UAVs and Fighter Aircraft*, *2*, 1998.

104. Trolletech, "Qt: C++ toolkit for multiplatform GUI & application development, version 3.2.3." `http://www.trolltech.com/qt/`.

105. Unknown. "Current and Future UAV Military Users and Applications," *AIR & SPACE EUROPE*, *1*(5 of 6):55 (1999).

106. USAF, Air Combat Command Public Affairs Office, "U.S. Air Force Fact Sheet: RQ-1 Predator Unmanned Aerial Vehicle," May 2002. `http://www.af.mil/news/factsheets/RQ_1_Predator_Unmanned_Aerial.html`; Department of the Air Force;.

107. Varga, Andras, "Objective Modular Network Testbed in C++." `http://www.hit.bme.hu/phd/vargaa/omnetpp/index.htm`.

108. Ware, Colin. *Information Visualization*. Interactive Technologies, San Francisco, California: Morgan Kaufmann, 2000.

109. Wilson, J.R. "UAVS: A Worldwide Roundup," *Aerospace America*, 1–10 (June 2003). `http://www.aiaa.org/aerospace/Article.cfm?issuetocid=365&ArchiveIssueID%=39`.

110. Yao, Yong and Johannes Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks." `citeseer.nj.nec.com/yao02cougar.html`.

111. Zbigniew Michalewicz, David B. Fogel. *How to Solve It: Modern Heuristics*. ACM Computing Classification, Springer-Verlag Bierline Heidelberg New York: Springer, 2002.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
23-03-2004	**Master's Thesis**	Aug 2003 – Mar 2004

4. TITLE AND SUBTITLE

SWARMING RECONNAISSANCE USING UNMANNED AERIAL VEHICLES IN A PARALLEL DISCRETE EVENT SIMULATION

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Corner, Joshua J., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/04-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFTA (AFMC)
Attn: Dr. Robert L. Ewing
2241 Avionics, Bldg 620, Rm S3-A52 DSN: 785-6635, ext. 3592
WPAFB, OH 45433 email: robert.ewing@wpafb.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Current military affairs indicate that future military warfare requires safer, more accurate, and more fault-tolerant weapons systems. Unmanned Aerial Vehicles (UAV) are one answer to this military requirement. Technology in the UAV arena is moving toward smaller and more capable systems and is becoming available at a fraction of the cost. Exploiting the advances in these miniaturized flying vehicles is the aim of this research. How are the UAVs employed for the future military? The concept of operations for a micro-UAV system is adopted from nature from the appearance of flocking birds, movement of a school of fish, and swarming bees among others. All of these natural phenomena have a common thread: a global action resulting from many small individual actions. This "emergent behavior" is the aggregate result of many simple interactions occurring within the flock, school, or swarm. In a similar manner, a more robust weapon system uses emergent behavior resulting in no "weakest link" because the system itself is made up of simple interactions by hundreds or thousands of homogeneous UAVs. The global system in this research is referred to as a swarm. Losing one or a few individual unmanned vehicles would not dramatically impact the "swarms" ability to complete the mission or cause harm to any human operator. Swarming reconnaissance is the emergent behavior of swarms to perform a reconnaissance operation. An in-depth look at the design of a reconnaissance swarming mission is studied. A taxonomy of passive reconnaissance applications is developed to address feasibility. Evaluation of algorithms for swarm movement, communication, sensor input/analysis, targeting, and network topology result in priorities of each model's desired features. After a thorough selection process of available implementations, a subset of those models are integrated and built upon resulting in a simulation that explores the innovations of swarming UAVs. Visualization of the swarm is accomplished through a post-processing visual system as well as a near real-time system. Exploration of these concepts is accomplished through a high performance computing parallel discrete event simulation. That platform is used as the test bed for swarming reconnaissance.
After development of the system, several experiments are designed, tested, and analyzed for efficiency and effectiveness. Results indicate that swarming reconnaissance is a feasible option for our future military.

15. SUBJECT TERMS

Unmanned, Aerial Reconnaissance, Aerial Warfare, Formation Flight, Discrete Event Simulation, High Performance Computing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Gary B. Lamont, AD-24, DAF
U	U	U	UU	187	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, ext 4718; e-mail: gary.lamont@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18